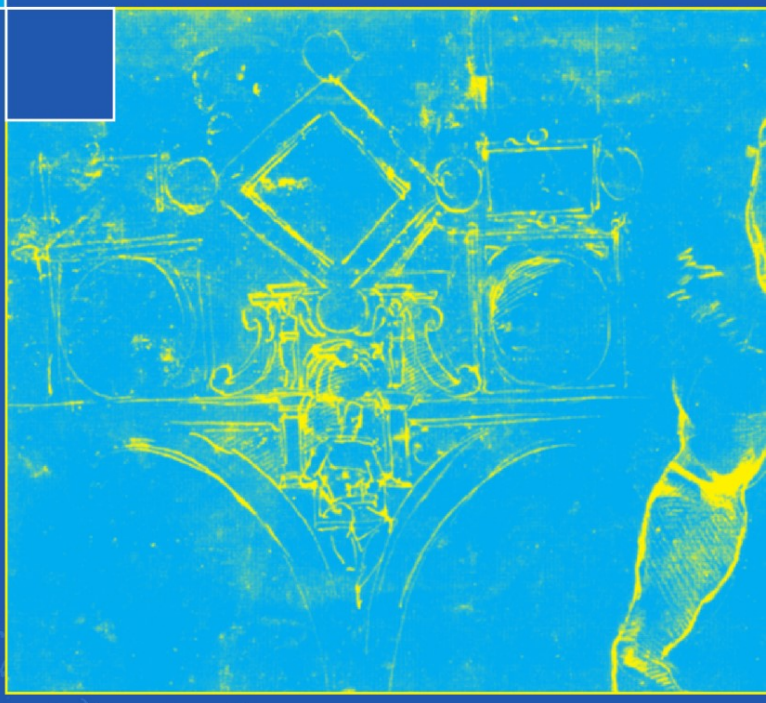
Vijay V. Vazirani

Collection IRIS
dirigée par Nicolas Puech



Algorithmes d'approximation

Traduction : Nicolas Schabanel

Springer

Algorithmes d'approximation

Springer

Paris
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Tokyo

Vijay V. Vazirani

Algorithmes d'approximation

Traduit de l'anglais par Nicolas Schabanel

Springer

Vijay V. Vazirani Ph. D.
Georgia Institute of Technology
College of Computing
801 Atlantic Avenue
Atlanta, GA 30332-0280
USA

Traducteur
Nicolas Shabanel
Chargé de Recherches au CNRS
Laboratoire de l'informatique du parallélisme
École normale supérieure de Lyon
46, avenue d'Italie
69364 Lyon Cedex 07
France

Traduit de l'édition anglaise :
Approximation Algorithms by Vijay V. Vazirani

ISBN : 3-540-65367-8

ISBN-10 : 2-287-00677-X Springer Paris Berlin Heidelberg New York
ISBN-13 : 978-2-287-00677-7 Springer Paris Berlin Heidelberg New York

Imprimé en France

Springer-Verlag France est membre du groupe Springer Science + Business Media

SPIN : 10918210

Maquette de couverture : Jean-François Montmarché
Mise en page : Nicolas Puech

À mes parents

« Cet ouvrage aborde les développements théoriques majeurs permettant d'obtenir des solutions approchées aux problèmes difficiles d'optimisation combinatoire ou de dénombrement. Il comporte une présentation élégante de la théorie combinatoire, de nombreux algorithmes aussi utiles qu'intéressants et des résultats remarquables concernant la complexité des problèmes traités. La clarté de l'exposé et la sélection judicieuse des exercices rendent cet ouvrage attractif et accessible pour tous les lecteurs ayant un goût pour les mathématiques et l'algorithmique. »

Richard Karp, Professeur à l'Université de Californie, Berkeley

« Suite au développement des techniques fondamentales d'optimisation combinatoire des années 1960 et 1970, une question importante consistait à développer une théorie algorithmique de l'approximation. Dans les années 1990, les progrès conjoints réalisés tant dans la conception d'algorithmes d'approximation que dans la découverte de preuves établissant la complexité de l'approximation de certains problèmes, ont conduit à une théorie élégante. La nécessité de résoudre des instances de plus en plus grandes pour des problèmes difficiles, comme ceux posés par Internet ou la génomique, a renforcé l'intérêt pour cette théorie. Il s'agit d'un domaine de recherche très actif qui voit sa boîte à outils s'enrichir de jour en jour.

C'est un plaisir de recommander l'ouvrage de Vijay Vazirani, complet et bien écrit, qui traite de ce sujet d'actualité. Je suis certain qu'il sera très utile au lecteur qui y trouvera aussi bien une introduction à l'algorithmique de l'approximation qu'un texte de référence sur de nombreux aspects du domaine. »

Lásló Lovász, Directeur de recherche, Microsoft Recherche

Préface

Bien que cela puisse paraître paradoxal, toute science exacte est dominée par la notion d'approximation.

Bertrand Russell (1872-1970)

La plupart des problèmes d'optimisation naturels sont **NP**-difficiles, et tout particulièrement ceux qui ont des applications réelles importantes. Suivant la conjecture largement admise $\mathbf{P} \neq \mathbf{NP}$, leur résolution exacte impliquerait un temps de calcul prohibitif. Caractériser la difficulté de l'approximation de ces problèmes par des algorithmes de temps polynomial est donc un sujet d'étude inévitable en informatique et en mathématiques. Cet ouvrage dresse un tableau de la théorie de l'algorithmique d'approximation à ce jour. Il est raisonnable de penser que cette image évoluera dans le temps.

L'exposé se divise en trois parties. La première partie présente des algorithmes d'approximation combinatoires pour des problèmes fondamentaux en mettant en œuvre une grande diversité de techniques algorithmiques. La diversité de ces techniques peut paraître déconcertante au premier abord. La nature est en effet riche et nous ne pouvons pas espérer qu'un nombre limité d'astuces permette de résoudre la très grande variété des problèmes **NP**-difficiles. Nous avons volontairement évité de trop catégoriser les différentes techniques, afin de ne pas restreindre leurs portées. Au contraire, nous avons tenté d'extraire aussi précisément que possible les caractéristiques individuelles de chaque problème, ainsi que les relations entre ces problèmes et les solutions algorithmiques proposées.

La deuxième partie est consacrée aux approximations reposant sur la programmation linéaire. Deux techniques fondamentales y sont présentées : la méthode de l'arrondi et la méthode primal-dual. La qualité de la solution approchée dépend principalement du programme linéaire choisi et de sa relaxation. Il n'y a pas de recette miracle pour trouver une bonne relaxation d'un problème, de même qu'il n'en existe pas pour démontrer un théorème en mathématiques (les lecteurs familiers avec la théorie de la complexité reconnaîtront la question sous-jacente $\mathbf{P} \neq \mathbf{NP}$).

La dernière partie détaille quatre thématiques importantes. La première est la recherche d'un vecteur non nul le plus court dans un module (chapitre 27). Différentes raisons font que ce problème mérite d'être traité à part.

La deuxième thématique étudie le dénombrement approché des solutions d'un problème, ce qui est une question très différente de celle de l'approximation de la valeur d'une solution d'un problème d'optimisation. Le dénombrement des solutions de presque tous les problèmes **NP**-complets connus est #**P**-complet. Il est intéressant de constater qu'à l'exception d'une poignée de problèmes, c'est également vrai pour les problèmes de **P**. Une théorie très avancée a été proposée pour obtenir des algorithmes de dénombrement efficaces pour les problèmes de **P**. La plupart de ces algorithmes utilisent des méthodes de Monte Carlo à base de chaînes de Markov (MCMC), un sujet qui nécessiterait un livre à lui tout seul et qui ne sera donc pas traité ici. Le chapitre 28 présente deux solutions algorithmiques combinatoires, qui n'utilisent pas la méthode MCMC, pour deux problèmes fondamentaux de dénombrement.

La troisième thématique est consacrée aux résultats récents qui ont confirmé le fondement d'une théorie propre de l'algorithmique d'approximation en quantifiant la difficulté de l'approximation pour plusieurs problèmes clés. Le chapitre 29 passe en revue ces résultats. Le principal point technique, le théorème PCP, y est admis. Nous ne connaissons malheureusement pas de preuve simple de ce théorème à l'heure actuelle.

Nous avons regroupé dans le dernier thème de nombreux problèmes ouverts de ce jeune champ de recherche. Cette liste n'est pas exhaustive. Elle est plutôt centrée autour des sujets d'étude actuellement actifs. On a cherché, durant quarante ans, des algorithmes exacts pour ces problèmes, sans réelle avancée. Étant donné que parmi les problèmes naturels, les algorithmes polynomiaux sont plutôt l'exception que la règle, il est raisonnable de croire que l'importance de l'algorithmique d'approximation va croître considérablement dans les années à venir.

Le problème de couverture par ensembles minimum occupe une place particulière dans la théorie de l'approximation et donc dans cet ouvrage. Sa définition très simple permet d'introduire à la fois des concepts clés et quelques techniques algorithmiques de base des parties I et II. La troisième partie complète le traitement de ce problème central par un résultat démontrant la difficulté de son approximation, dont la preuve est assez élaborée. La borne donnée par ce résultat est asymptotiquement égale à celle du meilleur algorithme d'approximation pour ce problème — ce qui est une raison supplémentaire de présenter cette preuve plutôt difficile.

Le travail du sculpteur Michel-Ange est un modèle pour notre approche de la conception et de la présentation des algorithmes. Une part essentielle du travail du sculpteur consiste en effet à étudier les pierres intéressantes d'une carrière, pour en découvrir les formes qu'elles épousent naturellement. Puis, le travail au ciseau consiste à suivre ces formes de manière naturelle, ou encore minimale. Par analogie, nous commençons par étudier un problème formulé

clairement et simplement (peut-être une version simplifiée du problème original). La plus grande partie du travail d'un algorithmicien est de comprendre quelle est la structure combinatoire déterminante sur le plan algorithmique. L'algorithme s'articule ensuite autour de cette structure de façon minimale. La présentation des algorithmes s'inscrit dans cette analogie, en insistant sur la découverte des structures des problèmes puis en présentant les algorithmes sous une forme minimale.

Nous avons essayé de rédiger chaque chapitre de façon concise et simple, quitte à ne présenter souvent que le résultat principal. Les généralisations et résultats similaires sont laissés en exercices. Nous avons également été amenés à proposer en exercices des résultats importants qu'il n'a pas été possible de développer en détail dans ce volume. Des indications sont données pour certains exercices sans pour autant que cela soit significatif de leur difficulté.

Cet ouvrage correspond à un ou plusieurs cours d'algorithmes d'approximation de fin de licence ou de mastère. Il contient plus du double de ce qui peut être présenté en un semestre, permettant ainsi à l'enseignant de choisir librement les sujets abordés. Un cours de licence d'introduction à l'algorithmique, à la théorie des graphes et sur la théorie de la **NP**-complétude devrait être un prérequis suffisant pour la plupart des chapitres. Par souci de complétude, le lecteur trouvera en annexe divers rappels des principaux résultats et définitions utilisés : théorie de la complexité en annexe A, probabilités en annexe B, programmation linéaire au chapitre 12, programmation semi-définie au chapitre 26, modules au chapitre 27. Nous avons choisi d'accorder une part sans doute disproportionnée à l'autoréductibilité dans l'annexe A car cette notion est rarement traitée ailleurs. Cet ouvrage peut également compléter un cours d'algorithmique fondamentale en 1^{er} ou $2^{\text{ème}}$ cycle (tout particulièrement les premiers chapitres des parties I et II). Les chapitres de ces deux parties sont de difficulté croissante.

Afin de toucher un public aussi large que possible, nous avons décidé de ne publier cet ouvrage dans aucune des séries spéciales de Springer — pas même la prestigieuse série jaune (nous n'avons cependant pas pu résister à l'ajout d'une touche de jaune sur la couverture de l'édition originale). Tous les commentaires et toutes les corrections sont les bienvenus. Nous avons créé une adresse électronique à cet effet pour l'édition en langue française : algo_approx@gmail.com[1].

Enfin, quelques mots à propos de l'impact concret de ces résultats. Sachant qu'en pratique les gens ont les yeux rivés sur des marges d'erreur entre 2 % et 5 %, quelle est l'utilité d'algorithmes à un facteur 2, voire à $O(\log n)$ de l'optimal ? De même, quel est l'intérêt d'améliorer la qualité de l'approximation d'un facteur 2 à 3/2 ?

Considérons ces deux questions en explicitant tout d'abord leur caractère fallacieux. La garantie de performance est une borne sur la performance

[1] L'adresse électronique dédiée aux remarques concernant l'édition originale est : approx@cc.gatech.edu.

de l'algorithme pour la pire instance. Il vaut peut-être mieux voir le facteur d'approximation comme une mesure qui encourage à explorer plus en détails la structure combinatoire du problème, afin de découvrir des outils plus puissants pour l'exploiter. Il est reconnu qu'il est d'autant plus difficile de trouver des instances critiques atteignant les bornes que les algorithmes offrent de meilleures garanties. Pour certains algorithmes récents, l'obtention de telles instances a fait l'objet d'un article à elle seule (par exemple, voir section 26.7). Des expérimentations ont confirmé que ces algorithmes sophistiqués ont la marge d'erreur souhaitée, de 2 % à 5 %, sur les instances typiques rencontrées en pratique, même si leur pire cas est bien au-delà. Enfin, les algorithmes prouvés théoriquement doivent être considérés comme une base algorithmique qu'il faut adapter soigneusement aux instances se présentant dans des applications spécifiques.

Nous espérons que ce livre catalysera la croissance de cette théorie et de son impact en pratique.

Remerciements

Ce livre s'appuie sur les cours que j'ai donnés à l'Indian Institute of Technology de Delhi durant les sessions de printemps 1992 et 1993, à Georgia Tech durant les sessions de printemps 1997, 1999 et 2000 et à DIMACS durant l'automne 1998. Les notes de cours du printemps 1992 constituèrent la première pierre de cet ouvrage. Il est intéressant de remarquer que plus de la moitié de notre propos repose sur des résultats de recherche issus des années qui ont suivi.

De très nombreux amis — et membres de ma famille — m'ont aidé. Je souhaite tout d'abord remercier Naveen Garg, Kamal Jain, Ion Măndoiu, Sridhar Rajagopalan, Huzur Saran et Mihalis Yannakakis — mes collaborations intenses avec eux m'ont permis de formuler les idées présentées dans cet ouvrage. J'ai eu la chance de bénéficier de l'aide et des conseils de Ion Măndoiu sur de nombreux points — ses qualités artistiques furent d'un grand secours pour la présentation du document et de ses figures. Merci encore, Ion !

Je souhaite exprimer ma gratitude envers les très nombreux experts du domaine pour leur aide généreuse sur le choix des thèmes à aborder et leur présentation, pour leurs commentaires sur les manuscrits, pour avoir assuré la correction et la complétude des références, ainsi que pour la conception des exercices et de la liste des problèmes ouverts. Merci à Sanjeev Arora, Alan Frieze, Naveen Garg, Michel Goemans, Mark Jerrum, Claire Kenyon, Samir Khuller, Daniele Micciancio, Yuval Rabani, Sridhar Rajagopalan, Dana Randall, Tim Roughgarden, Amin Saberi, Leonard Schulman, Amin Shokrollahi, et Mihalis Yannakakis. Un merci tout particulier à Kamal Jain, Éva Tardos, et Luca Trevisan.

De nombreuses autres personnes m'ont aidé par la pertinence de leurs commentaires. Je souhaite remercier Sarmad Abbasi, Cristina Bazgan, Ro-

gerio Brito Gruia Calinescu, Amit Chakrabarti, Mosses Charikar, Joseph Cheriyan, Vasek Chvátal, Uri Feige, Cristina Fernandes, Ashish Goel, Parikshit Gopalan, Mike Grigoriadis, Sudipto Guha, Dorit Hochbaum, Howard Karloff, Leonid Khachian, Stavros Kolliopoulos, Jan van Leeuwen, Nati Lenial, George Leuker, Vangelis Markakis, Aranyak Mehta, Rajeev Motwani, Prabhakar Raghavan, Satish Rao, Miklos Santha, Jiri Sgall, David Shmoys, Alistair Sinclair, Prasad Tetali, Pete Veinott, Ramarathnam Venkatesan, Nisheeth Vishnoi, et David Williamson. J'adresse mes excuses et mes remerciements aux personnes que j'aurais oubliées. De nombreux étudiants ont joué un rôle important, notamment en me confiant les notes qu'ils avaient prises dans mes cours. Je voudrais leur exprimer ma gratitude.

Je souhaite remercier l'IIT Delhi — remerciements particuliers à Shachin Maheshwari — Georgia Tech et DIMACS pour m'avoir fourni des environnements de travail agréables, motivants et riches. Merci à la NSF pour ses dotations financières CCR-9627308 et CCR-9820896.

J'ai particulièrement apprécié de travailler avec Hans Wössner à l'édition de ce livre. Le soin qu'il accorda à ce projet et à son auteur furent impressionnants. Merci encore à Frank Holzwarth pour avoir partagé son expertise en LaTeX.

Un projet de cette importance aurait été difficile à mener à bien sans le soutien sans réserve des membres de ma famille. Heureusement pour moi, quelques-uns sont également des chercheurs — ma femme, Milena Mihail, et mon frère, Umesh Vazirani. L'arrivée de Little Michel à mi-chemin de ce projet m'a apporté de nouvelles joies et énergies, et a rendu ce projet encore plus audacieux ! Je souhaite remercier avant tout mes parents pour leur soutien indéfectible et leur inspiration — mon père, auteur distingué de plusieurs ouvrages d'ingénierie civile, et ma mère, qui connaît si bien la musique classique indienne. Ce livre leur est dédicacé.

Atlanta, Géorgie, mai 2001 Vijay Vazirani

Remerciements du traducteur

Je tiens à remercier Vincent Bouchitté, Maxime Crochemore, Marianne Delorme, Marc Demange, Paul Feautrier, Fabien Feschet, Pierre Fraigniaud, Cyril Gavoille, Isabelle Guérin-Lassous, Olivier Hudry, Claire Kenyon, Pascal Koiran, Emmanuelle Lebhar, Laurent Lyaudet, Jacques Mazoyer, Alain Maruani, Jérôme Monnot, Bruno Petazzoni, Nicolas Puech, Mathieu Raffinot, André Raspaud, Guillaume Theyssier, Arnaud Tisserand, Ioan Todinca, et bien d'autres que j'aurais pu oublier, pour leurs aides, relectures et encouragements lors de cette traduction.

Lyon, France, novembre 2005 Nicolas Schabanel

Sommaire

1 Introduction

Les problèmes d'optimisation **NP**-difficiles forment un ensemble très riche de possibilités : de la possibilité d'approcher avec une précision arbitraire, à l'impossibilité de toute garantie sur la qualité de l'approximation. Malgré cette diversité, la conception d'algorithmes d'approximation suit des principes généraux, que nous allons explorer dans ce chapitre.

Un problème d'optimisation ne se résout en temps polynomial que s'il possède une structure combinatoire pertinente algorithmiquement, qui offre une « prise » pour trouver efficacement une solution optimale. La conception d'un algorithme polynomial exact s'exécute en deux phases : démêler une structure utile du problème, puis trouver un algorithme qui puisse l'exploiter.

Bien que les problèmes **NP**-difficiles n'offrent pas de prises pour trouver une solution optimale efficacement, ils peuvent parfois offrir des prises pour trouver une solution quasi optimale. Ainsi, la conception d'algorithmes d'approximation n'est pas, à première vue, très différente de la conception d'algorithmes exacts. Il s'agit également de démêler une structure pertinente du problème et de trouver un algorithme pour l'exploiter efficacement. Mais, en général, cette structure est plus élaborée, et souvent les techniques algorithmiques utilisées sont obtenues par généralisation ou extension de plusieurs outils puissants développés pour l'étude des algorithmes exacts.

La conception d'algorithmes d'approximation suit cependant ses propres principes. Nous illustrons quelques-uns de ces principes dans la section 1.1, pour le problème suivant.

Problème 1.1 (Couverture par sommets)[1] Étant donné un graphe non orienté $G = (V, E)$ et une fonction de coût sur les sommets $c : V \to \mathbf{Q}^+$, trouver une *couverture par sommets de coût minimal*, c'est-à-dire un sous-ensemble $S \subseteq V$ tel que toutes les arêtes ont (au moins) une extrémité dans S. Le cas particulier où tous les sommets ont un coût unitaire sera appelé le problème de la *couverture par sommets de taille minimum*.[2]

La conception d'algorithmes d'approximation implique une attaque délicate de la **NP**-difficulté afin d'en extraire une solution approchée efficacement. L'annexe A et plusieurs exercices de la section 1.3 présentent des rappels utiles sur les concepts clés de la théorie de la complexité.

[1] *Vertex cover*, en anglais.
[2] *Cardinality vertex cover problem*, en anglais.

Les définitions précises des notions de problème d'optimisation **NP** et d'algorithme d'approximation qui le résout (par exemple, voir exercices 1.9 et 1.10) sont importantes mais techniques ; nous avons donc préféré les donner en annexe A. Le lecteur en retrouvera cependant l'essentiel ci-dessous.

Un problème d'optimisation **NP**, Π, est un problème de minimisation ou de maximisation. À chaque instance valide I de Π, est associé un ensemble de solutions réalisables. À chaque solution réalisable est associé un nombre rationnel, appelé sa *valeur objectif*, ou encore la valeur de la *fonction objectif* en cette solution. Pour un problème d'optimisation **NP**, il existe des algorithmes polynomiaux (en la taille du codage de I et de la solution testée) qui décident de la validité et de la réalisabilité d'une solution et qui en calcule la valeur objectif. Une solution réalisable qui a la valeur optimale de la fonction objectif est appelée *solution optimale*. $\mathrm{OPT}_\Pi(I)$ désigne la valeur objectif d'une solution optimale d'une instance I. Nous l'abrégerons par OPT en absence d'ambiguïté. Le calcul de $\mathrm{OPT}_\Pi(I)$, pour les problèmes étudiés dans cet ouvrage, est **NP**-difficile.

Par exemple, une instance valide pour la couverture par sommets consiste en un graphe non orienté $G = (V, E)$ muni d'une fonction de coût sur ses sommets. Une solution réalisable est un sous-ensemble $S \subseteq V$ qui couvre les arêtes G. Sa valeur objectif est la somme des coûts des sommets de S. Un tel ensemble de coût minimum est une solution optimale.

Un algorithme d'approximation $\mathcal{A}$ pour Π calcule en temps polynomial (en la taille de l'instance) une solution réalisable dont la valeur objectif est « proche » de l'optimal ; par « proche », on entend que cette valeur est à un facteur garanti de la valeur optimale. Dans la section suivante, nous présentons un algorithme polynomial de facteur d'approximation 2 pour le problème de la couverture par sommets de taille minimum, c'est-à-dire un algorithme qui trouve une couverture par sommets de coût inférieur à $2 \cdot \mathrm{OPT}$ en temps polynomial en $|V|$ (le nombre d'arêtes, et donc la taille d'une instance, sont des polynômes en $|V|$). Un tel algorithme polynomial sera appelé une 2-approximation.

1.1 Minorer OPT

La conception d'un algorithme d'approximation pour un problème d'optimisation **NP** et **NP**-difficile fait face immédiatement au dilemme suivant. Le coût de la solution produite doit être comparé au coût optimal afin d'établir la garantie de performance. Or, pour ces problèmes, non seulement le calcul d'une solution optimale est **NP**-difficile, mais le calcul du coût optimal l'est aussi (voir annexe A). Nous verrons en fait, section A.5, que le calcul du coût optimal (ou la résolution du problème de décision associé) est au cœur de la résolution de ces problèmes. Comment alors établir la garantie de performance ? Il est intéressant de noter que la réponse à cette question est une étape clé de la conception d'algorithmes d'approximation.

Prenons l'exemple de la couverture par sommets de taille minimum. Nous contournons la difficulté ci-dessus en construisant un « bon » *minorant*[3] du coût optimal (la taille minimum d'une couverture par sommets), qui, lui, sera calculable en temps polynomial.

1.1.1 Un algorithme d'approximation pour la couverture par sommets de taille minimum

Commençons par quelques définitions. Étant donné $H = (U, F)$, un graphe, on appelle *couplage* de H, tout sous-ensemble d'arêtes $M \subseteq F$ tel qu'aucune paire d'arêtes de M n'a d'extrémité commune. On appelle *couplage maximum*, tout couplage de taille maximum dans H, et *couplage maximal*, tout couplage qui est maximal pour l'inclusion[4]. L'algorithme glouton suivant calcule en temps polynomial un couplage maximal : tant qu'il existe des arêtes, sélectionner une arête et retirer ses extrémités du graphe. Des algorithmes plus sophistiqués permettent également de calculer un couplage maximum en temps polynomial.

Observons que la taille d'un couplage maximal de G est un minorant pour la couverture par sommets de taille minimum. La raison en est que *toute* couverture par sommets doit contenir au moins l'une des extrémités de chaque arête du couplage. La construction de ce minorant suggère immédiatement l'algorithme très simple suivant :

Algorithme 1.2 (Couverture par sommets de taille minimum)

Calculer un couplage maximal de G et renvoyer l'ensemble des extrémités des arêtes du couplage.

Théorème 1.3 *L'algorithme 1.2 est une 2-approximation pour le problème de la couverture par sommets de taille minimum.*

Preuve : Toutes les arêtes sont couvertes par l'ensemble de sommets renvoyé — sinon, une arête non couverte pourrait être ajoutée au couplage, qui ne serait pas maximal. Soit M le couplage sélectionné. Nous savons que $|M| \leqslant \mathrm{OPT}$. Le facteur d'approximation se déduit donc ainsi : la taille de la couverture renvoyée est $2\,|M|$, c'est-à-dire $\leqslant 2 \cdot \mathrm{OPT}$. □

[3] *A good lower bound*, en anglais.

[4] Les termes « minimum » et « minimal » réfèrent à deux notions distinctes sur les ordres : un élément minimum d'un ordre est un élément qui est inférieur à tous les éléments de l'ordre, alors qu'un élément minimal est un élément tel qu'aucun autre élément ne lui soit inférieur. Un élément minimum est en particulier minimal, mais la réciproque n'est pas vraie en général (par exemple, elle est vraie pour les ordres totaux). La même remarque s'applique aux termes « maximum » et « maximal ».

Notons que l'algorithme d'approximation découle naturellement de la construction du minorant. Cette situation est typique en algorithmique d'approximation. Nous verrons, partie II, comment la programmation linéaire permet d'unifier la construction de minorants de plusieurs problèmes fondamentaux ; les algorithmes d'approximation se construisent alors directement à partir du programme linéaire qui calcule le minorant.

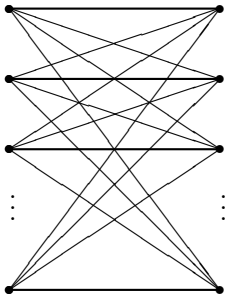
1.1.2 Peut-on améliorer la garantie de performance ?

Plusieurs questions se posent lorsque l'on cherche à améliorer la garantie de performance pour la couverture par sommets de taille minimum :

1. Peut-on améliorer le facteur d'approximation de l'algorithme 1.2 par une meilleure analyse ?
2. Peut-on construire un algorithme avec une meilleure garantie en utilisant le même minorant que l'algorithme 1.2, c'est-à-dire la taille d'un couplage maximum de G ?
3. Existe-t-il une autre construction de minorant qui améliore le facteur d'approximation pour la couverture par sommets ?

L'exemple 1.4 montre que la réponse à la première question est « Non », c'est-à-dire l'analyse de l'algorithme 1.2 donnée ci-dessus est *au plus près*. Il donne une famille infinie d'instances pour lesquelles le coût de la solution proposée par l'algorithme 1.2 est le double de l'optimal. Une famille infinie d'instances de ce type, montrant que l'analyse d'un algorithme d'approximation est au plus près, sera appelée une *instance critique*.[5] La construction d'instances critiques est particulièrement importante. Ces dernières permettent de mieux comprendre les faiblesses d'un algorithme et ont souvent permis d'obtenir des algorithmes avec de meilleures garanties. Nous conseillons vivement au lecteur d'exécuter les algorithmes de ce livre sur leurs instances critiques.

Exemple 1.4 Considérons la famille infinie d'instances formées des graphes bipartis complets $K_{n,n}$.



[5] *Tight analysis* et *tight instance*, en anglais.

L'algorithme 1.2 sélectionne l'ensemble des $2n$ sommets de $K(n,n)$, alors que chaque côté du graphe biparti forme une couverture de taille n. □

Supposons que les facteurs d'approximation soient toujours analysés en comparant le coût de la solution calculée au minorant. C'est en fait le cas de la plupart des algorithmes d'approximation connus. Alors, la réponse à la deuxième question est « Non » également. L'exemple 1.5 exhibe une famille infinie d'instances pour lesquelles le minorant, la taille d'un couplage maximal, vaut la moitié de la taille de la couverture minimale. Dans le cas d'algorithmes fondés sur un programme linéaire, la question analogue se pose en terme d'une quantité fondamentale en programmation linéaire, le saut intégral[6] (voir chapitre 12).

La troisième et dernière question, améliorer la garantie de performance pour la couverture par sommets, est en fait un problème ouvert central de l'algorithmique d'approximation actuellement (voir section 30.1).

Exemple 1.5 Pour la famille infinie d'instances $(K_n)_{n \text{ impair}}$ (les cliques de taille impaire), le minorant, la taille d'un couplage maximal, vaut la moitié de la taille minimale d'une couverture par sommets. Tout couplage maximal compte exactement $(n-1)/2$ arêtes, alors que la taille d'une couverture minimale est $n-1$. □

1.2 Problèmes bien caractérisés et relations min-max

Considérons les problèmes de décision associés aux problèmes de la couverture par sommets de taille minimum et du couplage maximum :

– la taille d'une couverture minimum de G est-elle $\leqslant k$?
– la taille d'un couplage maximum de G est-elle $\geqslant \ell$?

Ces deux problèmes de décision sont dans **NP** et ont donc des certificats positifs (définitions en annexe A). Ces problèmes ont-ils des certificats négatifs ? Nous savons que la taille d'un couplage maximum est un minorant de la taille d'une couverture par sommets minimum. Si G est biparti, on a en fait l'égalité ; c'est le théorème classique de König-Egerváry.

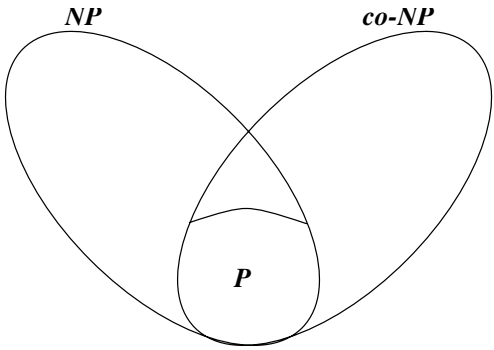
Théorème 1.6 *Pour tout graphe biparti,*

$$\max_{\textit{couplage } M} |M| = \min_{\textit{couverture par sommets } C} |C|$$

Ainsi, pour tout graphe G biparti, si la réponse au premier problème est « Non », il existe dans G un couplage de taille $k+1$ qui est un certificat. De même, si la réponse au second problème est « Non », il existe une couverture de taille $\ell - 1$ dans G. Par conséquent, les restrictions de ces deux problèmes aux graphes bipartis admettent des certificats négatifs et appartiennent à

[6] *Integrality gap*, en anglais.

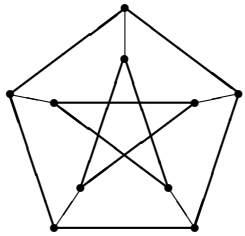
co-**NP**. En fait, ces deux restrictions sont dans **P**. Il est facile de voir que tout problème de **P** admet des certificats positifs et négatifs (la chaîne vide suffit). C'est l'inclusion suivante : $\mathbf{P} \subseteq \mathbf{NP} \cap \text{co-}\mathbf{NP}$. Le consensus actuel estime que cette inclusion est stricte ; cette conjecture est représentée ci-dessous.



Les problèmes qui ont des certificats positifs et négatifs, c'est-à-dire qui sont dans $\mathbf{NP} \cap \text{co-}\mathbf{NP}$, sont dits *bien caractérisés*. Cette notion est importante ; par exemple, c'est en remarquant que le problème du couplage maximum est bien caractérisé que la quête d'un algorithme polynomial pour ce problème a commencé.

Des relations min-max de ce type permettent de prouver qu'un problème est bien caractérisé. Ces relations sont parmi les résultats les plus élégants et les plus puissants de la combinatoire. Plusieurs algorithmes polynomiaux (exacts) fondamentaux en ont découlé. En fait, ces relations min-max sont, pour la plupart, des cas particuliers de la théorie de la dualité en programmation linéaire (voir section 12.2). Comme nous l'avons signalé plus haut, la dualité en programmation linéaire joue également un rôle fondamental dans la conception d'algorithmes d'approximation.

Que dire si G n'est plus biparti ? Dans ce cas, un couplage maximum peut être strictement plus petit que la couverture par sommets minimum. Par exemple, si G est un cycle de longueur impaire $2p+1$, la taille d'un couplage maximum est p, alors qu'une couverture par sommets optimale compte $p+1$ sommets. L'inégalité peut être stricte, même si le graphe admet un couplage parfait ; par exemple, le graphe de Petersen ci-dessous. Ce graphe admet un couplage parfait de taille 5 ; mais la taille d'une couverture par sommets minimum est 6. Ce graphe n'admet pas de couverture par sommets de taille 5 car toute couverture doit choisir au moins $p+1$ sommets de tout cycle de longueur impaire $2p+1$, pour couvrir les arêtes de ce cycle, et le graphe de Petersen a deux cycles disjoints de longueur 5.



Selon la conjecture **NP** $\neq$ co-**NP**, les problèmes **NP**-difficiles n'admettent pas de certificats négatifs. Ainsi, le problème de la couverture par sommets pour les graphes généraux, qui est **NP**-difficile, n'a-t-il pas de certificats négatifs, à moins que **NP** = co-**NP**. En revanche, le problème du couplage maximum pour les graphes généraux est dans **P**. Les certificats négatifs pour ce problème ne sont pas des couvertures par sommets, mais une structure plus générale : les couvertures impaires.

Une *couverture impaire* C d'un graphe $G = (V, E)$ est formée d'une suite de sous-ensembles $S_1, \ldots, S_k$ de cardinal impair de V et d'une suite $v_1, \ldots, v_\ell$ de sommets, telles que pour toute arête e de G, e a pour extrémité l'un des v_i, ou bien les deux extrémités de e appartiennent toute deux à l'un des S_j. Le *poids* de cette couverture C est défini par $w(C) = \ell + \sum_{i=1}^{k} (|S_i| - 1)/2$. Nous avons alors la relation min-max suivante :

Théorème 1.7 *Pour tout graphe,*

$$\max_{\textit{couplage } M} |M| = \min_{\textit{couverture impaire } C} w(C)$$

En général un couplage maximum peut être strictement plus petit qu'une couverture par sommets de taille minimum. Peut-il être arbitrairement plus petit ? La réponse est « Non ». Un corollaire du théorème 1.3 est que dans tout graphe la taille d'un couplage maximum est au moins la moitié de celle d'une couverture minimum. Plus précisément, le théorème 1.3 induit l'inégalité min-max suivante. Les algorithmes d'approximation conduisent fréquemment à ce genre d'inégalités min-max, toutes aussi intéressantes.

Corollaire 1.8 *Dans tout graphe,*

$$\max_{\textit{couplage } M} |M| \leqslant \min_{\textit{couverture par sommets } U} |U| \leqslant 2 \cdot \max_{\textit{couplage } M} |M|$$

Bien que le problème de la couverture par sommets n'admette pas de certificat négatif (à moins que **NP** = co-**NP**), il existe un moyen de certifier que (G, k) est une instance négative pour des valeurs suffisamment petites de k. L'algorithme 1.2 (plus précisément le minorant utilisé) apporte une réponse. Notons $\mathcal{A}(G)$ la taille de la couverture générée par l'algorithme 1.2. Alors, $\mathrm{OPT}(G) \leqslant \mathcal{A}(G) \leqslant 2 \cdot \mathrm{OPT}(G)$. Si $k < \mathcal{A}(G)/2$ alors $k < \mathrm{OPT}(G)$, et donc (G, k) est une instance négative. Ainsi, l'algorithme 1.2 produit un certificat négatif pour chaque instance (G, k) telle que $k < \mathrm{OPT}(G)/2$.

Un certificat négatif pour des instances (I, B) d'un problème de minimisation Π, telles que $B < \mathrm{OPT}(I)/\alpha$ est appelé un *certificat négatif approché de facteur* α. De même que pour les certificats positifs et négatifs habituels, ce certificat n'est pas nécessairement calculable en temps polynomial. Une α-approximation $\mathcal{A}$ pour Π produit un tel certificat, et puisque $\mathcal{A}$ est un algorithme polynomial, ce certificat est calculable en temps polynomial. Nous découvrirons au chapitre 27 un résultat fascinant : le problème du vecteur non nul le plus court d'un module admet un certificat négatif approché de facteur n ; mais on ne connaît aucun algorithme polynomial pour en construire.

1.3 Exercices

1.1 Proposez une 1/2-approximation pour le problème suivant :

Problème 1.9 (Sous-graphe sans cycle maximum) Étant donné un graphe orienté $G = (V, E)$, trouvez un sous-ensemble d'arêtes sans cycle et de taille maximum.
Indication : Numérotez arbitrairement les sommets de 1 à $|V|$ puis sélectionnez le plus grand des deux ensembles d'arêtes suivants : l'ensemble des arêtes croissantes (i, j) où $i < j$, et son complémentaire. Quelle méthode utilisez-vous pour majorer OPT ?

1.2 Proposez une 2-approximation pour trouver un couplage maximal de taille minimum dans un graphe non orienté.
Indication : Un couplage maximal est de taille au moins égale à la moitié de celle d'un couplage maximum.

1.3 (R. Bar-Yehuda) Voici une 2-approximation pour le problème de la couverture par sommets de taille minimum. Construisez un arbre de parcours en profondeur du graphe G, et indiquez l'ensemble S de tous les sommets internes de cet arbre. Démontrez que S couvre bien les arêtes de G et que $|S| \leqslant 2 \cdot \mathrm{OPT}$.
Indication : Démontrez que G possède un couplage de taille $\geqslant \lceil |S|/2 \rceil$.

1.4 Pour résoudre de façon approchée un problème d'optimisation, le premier réflexe est sans doute de proposer un algorithme glouton. Pour le problème de la couverture par sommets, un tel algorithme sélectionnerait un sommet de degré maximum, supprimerait ce sommet du graphe ainsi que ses arêtes incidentes et itérerait jusqu'à ce que le graphe n'ait plus d'arête. Démontrez que cet algorithme est une $O(\log n)$-approximation. Donnez une instance critique.
Indication : L'analyse est similaire à celle du théorème 2.4.

1.5 L'algorithme glouton suivant calcule un couplage maximal : sélectionner une arête, retirer ses deux extrémités du graphe et itérer jusqu'à ce que le graphe soit totalement déconnecté. Cela fait-il de l'algorithme 1.2 un algorithme glouton ?

1.6 Donnez un minorant pour la couverture par sommets de coût minimum.
Indication : Difficile sans utiliser la dualité en programmation linéaire.

1.7 Soit $A = \{a_1, \ldots, a_n\}$ un ensemble fini, et $\preccurlyeq$ un *ordre partiel* sur A, c'est-à-dire une relation réflexive, transitive et antisymétrique. Deux éléments a_i et a_j de A sont dits *comparables* si $a_i \preccurlyeq a_j$ ou $a_j \succcurlyeq a_i$, et *incomparables* sinon. Une *chaîne* est un sous-ensemble d'éléments de A deux à deux comparables. Une *antichaîne* est un sous-ensemble d'éléments de A deux à deux incomparables. Une *couverture par chaînes* (resp. *antichaînes*) de A est une partition de A en chaînes (resp. antichaînes). La taille d'une telle couverture est le nombre de chaînes (resp. antichaînes) qui la composent. Démontrez la relation min-max suivante : la taille (longueur) de la chaîne la plus longue est égale à la taille de la plus petite couverture par antichaînes.
Indication : Soit m la longueur de la chaîne la plus longue. Soit $\phi(a)$ la longueur de la chaîne la plus longue issue de a, c'est-à-dire constituée d'éléments plus grands que l'élément a. Considérez la partition de A par les $A_i = \{a \in A \mid \phi(a) = i\}$, pour $1 \leqslant i \leqslant m$.

1.8 (Théorème de Dilworth, voir [203]) Montrez que pour tout ordre partiel fini, la longueur de l'antichaîne maximum est égale à la taille minimum d'une couverture par chaînes.
Indication : Appliquez le théorème de König et Egerváry. Pour un ensemble partiellement ordonné A de taille n, considérez le graphe biparti $G = (U, V, E)$, où $|U| = |V| = n$ et $(u_i, v_j) \in E$ ssi $i \neq j$ et $a_i \preccurlyeq a_j$.

Les dix exercices suivants utilisent des notions abordées dans l'annexe A.

1.9 Le problème d'optimisation suivant est-il dans **NP** ? Soient $G = (V, E)$ un graphe non orienté muni d'une fonction de coût $c : V \to \mathbf{Q}^+$ sur les sommets et k un entier positif, trouver une couverture par sommets de coût minimal ayant au plus k sommets.
Indication : Peut-on décider de la validité d'une instance (c'est-à-dire si une instance admet au moins une solution réalisable) en temps polynomial ?

1.10 Soient Π un problème de minimisation **NP** et $\mathcal{A}$ un algorithme d'approximation randomisé pour Π, tel que pour toute instance, l'espérance du coût de la solution produite est $\leqslant \alpha \cdot \mathrm{OPT}$, avec $\alpha > 1$. Quel est le meilleur facteur d'approximation que l'on puisse garantir pour Π en utilisant $\mathcal{A}$?
Indication : Une garantie de $2\alpha - 1$ s'obtient facilement. Pour approcher arbitrairement le facteur α, exécutez l'algorithme un nombre polynomial de fois et sélectionnez la meilleure solution. Appliquez la borne de Chernoff.

1.11 Démontrez que si SAT est **NP**-difficile, et que s'il existe une réduction polynomiale de SAT au problème de la couverture par sommets de taille minimale, alors ce dernier est aussi **NP**-difficile.
Indication : Montrez que la composition de deux réductions polynomiales est également une réduction polynomiale.

1.12 Démontrez que si le problème de la couverture par sommets de taille minimale est dans co-**NP**, alors **NP** = co-**NP**.

1.13 (Pratt [231]) Démontrez que le langage des nombres premiers est dans **NP** ∩ co-**NP**.
Indication : Considérez le groupe multiplicatif des éléments inversibles modulo n, $Z_n^* = \{a \in Z_n \mid \text{pgcd}(a, n) = 1\}$. Clairement, $|Z_n^*| \leqslant n - 1$. Utilisez le fait que $|Z_n^*| = n - 1$ ssi n est premier, et que Z_n^* est cyclique si n est premier. Un certificat positif consiste en un générateur de Z_n^*, une factorisation de $n-1$ et récursivement, ces mêmes informations pour chaque facteur premier de $n-1$. Attention, vos algorithmes doivent être polynomiaux en $\log(n)$, la taille de n.

1.14 Prouvez que les problèmes d'optimisation présentés dans ce livre, sont autoréductibles : en particulier, le couplage maximum, MAX-SAT (problème 16.1), la clique maximum (problème 29.15), le surfacteur minimum (problème 2.9), et l'ordonnancement de temps d'exécution minimum (problème 10.1).
Indication : Pour la clique maximum, envisagez les deux possibilités : v est ou n'est pas dans la clique maximum ; et restreignez G à v et ses voisins ou retirez v de G, suivant le cas. Pour la surchaîne minimale, éliminez deux sous-chaînes et remplacez-les par une surchaîne valide (éventuellement la simple concaténation). Si la longueur de la surchaîne minimale est inchangée, alors continuez avec l'instance la plus courte. Généralisez légèrement le problème d'ordonnancement — considérez que l'instance du problème contient également le nombre d'unités de temps déjà ordonnancées sur chaque machine.

1.15 Proposez une définition d'autoréductibilité pour les problèmes **NP**, c'est-à-dire pour les problèmes de décision et non les problèmes d'optimisation, qui permette de trouver en temps polynomial une solution réalisable à partir d'un oracle pour le problème de décision associé, et qui, en sus, génère un arbre d'autoréductibilité pour chaque instance.
Indication : Ordonnez arbitrairement les « atomes »d'une solution, par exemple, pour SAT, ordonnez arbitrairement les n variables.

1.16 Soient Π_1 et Π_2 deux problèmes d'optimisation tels qu'il existe une réduction isofacteur[7] de Π_1 à Π_2. Montrez que s'il existe une α-approximation pour Π_2, il en existe une pour Π_1.

[7] *Approximation factor preserving reduction*, en anglais.

Indication : Commencez par montrer que si la réduction transforme une instance I_1 de Π_1 en une instance I_2 de Π_2 alors $\mathrm{OPT}_{\Pi_1}(I_1) = \mathrm{OPT}_{\Pi_2}(I_2)$.

1.17 Montrez que $L \in \mathbf{ZPP}$ ssi $L \in (\mathbf{RP} \cap \text{co-}\mathbf{RP})$.

1.18 Montrez que si $\mathbf{NP} \subseteq \text{co-}\mathbf{RP}$ alors $\mathbf{NP} \subseteq \mathbf{ZPP}$.
Indication : Si une instance ϕ de SAT est satisfaisable, alors on peut calculer avec forte probabilité une instanciation positive des variables en utilisant l'autoréductibilité et la machine co-**RP** pour SAT. Si ϕ n'est pas satisfaisable, la machine co-**RP** le confirme par une réponse négative, avec forte probabilité.

1.4 Notes

La notion de problème bien caractérisé a été introduite par Edmonds dans [76], puis a été formalisée précisément par Cook [54]. Dans le même article, Cook initia la théorie de la **NP**-complétude, découverte indépendamment par Levin [194]. Ces travaux prirent leur véritable valeur avec les travaux de Karp [172], démontrant que des problèmes algorithmiques fondamentaux très différents sont tous **NP**-complets.

Remarquons que les premiers algorithmes d'approximation ont été conçus avant la théorie de la **NP**-complétude : par Vizing [264] pour la coloration d'arêtes minimale, par Graham [120] (problème 10.1) pour minimiser le temps d'exécution d'un ordonnancement et par Erdös [80] pour MAX-CUT (problème 2.14). L'importance de concevoir de tels algorithmes s'est imposée à mesure que la conjecture $\mathbf{P} \neq \mathbf{NP}$ devenait plus vraisemblable. La définition d'algorithmes d'approximation a été posée formellement par Garey, Graham et Ullman [98] et Johnson [158]. Le premier algorithme à base de programmation linéaire a été conçu par Lovász [200] afin d'analyser l'algorithme glouton pour le problème de la couverture par ensembles (voir chapitre 13). Rabin [233], quant à lui, mena les tout premiers travaux exploratoires sur l'utilisation de la randomisation en algorithmique — cette notion est particulièrement utile en algorithmique d'approximation. Le théorème 1.7 fut prouvé par Edmonds [76] et l'algorithme 1.2 conçu indépendamment par Gavril et Yannakakis (voir [100]).

Nous conseillons la lecture des livres de Cormen, Leiserson, Rivest, et Stein [57], Papadimitriou et Steiglitz [226], et Tarpan [255] pour les techniques algorithmiques classiques. Le livre de Lovász et Plummer [203] propose un bon traitement des relations min-max. Pour les algorithmes d'approximation : Hochbaum [134] et Ausiello, Crescenzi, Gambosi, Kann, Marchetti, et Protasi [19]. Référez-vous aux sections 12.5, A.6, et B.4 pour la programmation linéaire, la théorie de la complexité, et les algorithmes randomisés, respectivement.

Première partie

Algorithmes combinatoires

2 Couverture par ensembles

Le problème de la couverture par ensembles est à l'algorithmique d'approximation ce que le problème du couplage maximum est à l'algorithmique exacte : l'étude de ce problème est à l'origine des outils fondamentaux du domaine. Ce problème nous intéresse particulièrement car il propose un cadre très simple pour l'explication de nombreuses techniques algorithmiques de base. Ce chapitre présente deux techniques combinatoires : les algorithmes gloutons et la méthode du mille-feuille. Dans la deuxième partie de ce livre, nous présenterons, également sur ce problème, les méthodes d'arrondi et primal-dual en programmation linéaire. Du fait de sa généralité, le problème de la couverture par ensembles a de nombreuses applications parfois inattendues. Nous étudions dans ce chapitre une de ces applications — le problème du surfacteur minimum (le chapitre 7 décrira un algorithme plus performant).

Une des premières tentatives pour résoudre un problème d'optimisation est souvent d'écrire une sorte d'algorithme glouton. Même si cette stratégie est inopérante pour un problème donné, trouver des contre-exemples donne des indices cruciaux sur la structure du problème. De façon surprenante, l'algorithme glouton le plus simple et le plus naturel est essentiellement la meilleure approximation qu'on puisse espérer pour ce problème (reportez-vous au chapitre 29 pour la preuve formelle de ce résultat).

Le problème de la couverture par ensembles est sans doute l'application la plus naturelle de cette stratégie en algorithmique d'approximation. Nous présenterons par la suite la méthode du mille-feuille sur ce problème.

Problème 2.1 (Couverture par ensembles)[1] Étant donné un univers U de n éléments, une collection de sous-ensembles de U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, et une fonction de coût $c : \mathcal{S} \to \mathbf{Q}^+$, trouver un sous-ensemble de $\mathcal{S}$, de coût total minimum, qui couvre tous les éléments de U.

On appelle *fréquence* d'un élément de U, le nombre de sous-ensembles auxquels il appartient. La fréquence f de l'élément le plus fréquent est un paramètre utile du problème. Les différents algorithmes pour la couverture par ensembles obtiennent l'un ou l'autre des facteurs d'approximation suivants : $O(\log n)$ ou f. Clairement, aucun des deux ne domine l'autre pour toutes les instances. Le problème restreint au cas où $f = 2$ est équivalent au

[1] *Set cover*, en anglais.

problème de la couverture par sommets (voir exercice 2.7), pour lequel nous avons donné une 2-approximation au chapitre 1.

2.1 L'algorithme glouton

Voici l'algorithme : choisir l'ensemble de coût efficace courant minimum, éliminer les éléments couverts et itérer jusqu'à ce que tous les éléments soient couverts. Soit C l'ensemble des éléments déjà couverts au début d'une itération. Le *coût efficace* d'un ensemble S, durant cette itération, est le prix payé, si S est sélectionné, pour couvrir chacun des nouveaux éléments couverts, c'est-à-dire $c(S)/|S \setminus C|$. Le *prix* d'un élément est le coût efficace payé pour le couvrir. De façon équivalente, on peut imaginer que, quand un ensemble S est choisi, son coût est réparti uniformément parmi les nouveaux éléments ajoutés, pour établir leur prix.

Algorithme 2.2 (Algorithme glouton pour la couverture par ensembles)

1. $C \leftarrow \varnothing$.
2. Tant que $C \neq U$ faire
   Soit S, l'ensemble de coût efficace minimum.
   Soit $\alpha = \frac{c(S)}{|S \setminus C|}$, le coût efficace de S.
   Sélectionner S et pour chaque $e \in S \setminus C$, poser $\text{prix}(e) = \alpha$.
   $C \leftarrow C \cup S$.
3. Renvoyer les ensembles sélectionnés.

Numérotons $e_1, \ldots, e_n$, les éléments de U dans l'ordre dans lequel ils sont couverts par l'algorithme (les *ex æquo* sont classés arbitrairement).

Lemme 2.3 *Pour tout* $k \in \{1, \ldots, n\}$, $\text{prix}(e_k) \leq \text{OPT}/(n-k+1)$.

Preuve : À chaque itération, les ensembles non sélectionnés de la solution optimale couvrent les éléments restants pour un coût total inférieur à OPT. Or il reste $|\overline{C}|$ éléments à couvrir, où $\overline{C} = U \setminus C$. Il en existe donc un parmi eux dont le coût effectif est inférieur à $\text{OPT}/|\overline{C}|$. $\overline{C}$ contenait au moins $(n-k+1)$ éléments lorsque e_k a été couvert. Puisque e_k a été couvert par l'ensemble de coût effectif minimum à cette itération, il s'ensuit que :

$$\text{prix}(e_k) \leqslant \frac{\text{OPT}}{|\overline{C}|} \leqslant \frac{\text{OPT}}{n-k+1}.$$

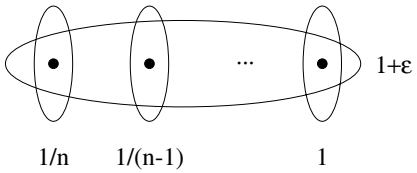
□

Le théorème suivant est une conséquence immédiate du lemme 2.3.

Théorème 2.4 *L'algorithme glouton est une H_n-approximation pour le problème de la couverture par ensembles minimum, avec $H_n = 1+\frac{1}{2}+\cdots+\frac{1}{n}$.*

Preuve : Le coût de chaque ensemble sélectionné est distribué sur les nouveaux éléments couverts. Le coût total de la couverture calculée est donc égal à $\sum_{k=1}^{n} \text{prix}(e_k)$. D'après le lemme 2.3, ce coût est $\leqslant \left(1+\frac{1}{2}+\cdots+\frac{1}{n}\right)\cdot\text{OPT}$. □

Exemple 2.5 Voici une instance critique pour l'algorithme 2.2 :



Sur cette instance, l'algorithme glouton produit la couverture constituée de n singletons car, à chaque itération, c'est un singleton qui a le coût effectif minimal. Ainsi, la couverture produite a pour coût :

$$\frac{1}{n}+\frac{1}{n-1}+\cdots+1 = H_n.$$

Or, la couverture optimale coûte $1+\varepsilon$. □

De façon surprenante, l'algorithme évident ci-dessus est essentiellement ce que l'on peut espérer de mieux pour le problème de la couverture par ensembles (voir sections 29.7 et 29.9).

Au chapitre 1, nous avons insisté sur le fait que trouver un bon minorant de OPT est une première étape pour la conception d'algorithmes d'approximation d'un problème de minimisation. Vous êtes en droit de vous interroger à ce stade de votre lecture sur la véracité de cette assertion. Nous verrons, section 13.1, que la bonne façon de voir l'algorithme glouton pour la couverture par ensembles s'inscrit dans le cadre de la théorie de la dualité en programmation linéaire — ce qui nous donnera non seulement le minorant sur lequel l'algorithme est bâti, mais aussi d'autres algorithmes pour des généralisations du problème.

2.2 La méthode du mille-feuille

Le problème de la couverture par ensembles est sans doute la meilleure introduction à la méthode du mille-feuille.[2] Remarquons cependant que cette technique n'est pas très générale. Nous donnons ici une 2-approximation

[2] *Layering*, en anglais.

pour la couverture par sommets munis de poids arbitraires. Nous laissons en exercice sa généralisation à une f-approximation pour la couverture par ensembles, où f est la fréquence maximale d'un élément (voir exercice 2.13).

L'idée sous-jacente du mille-feuille est de décomposer la fonction de poids sur les sommets en fonctions plus commodes, proportionnelles au degré, définies sur une suite de sous-graphes emboîtés de G. Pour de telles fonctions, nous allons démontrer que la solution obtenue en sélectionnant tous les sommets a un coût inférieur au double de coût optimum.

Commençons par quelques notations. Soit $w : V \to \mathbf{Q}^+$ la fonction de poids sur les sommets du graphe $G = (V, E)$. Une fonction de poids sur les sommets est dite *par degré* s'il existe une constante $c > 0$ telle que le poids de chaque sommet $v \in V$ est $c \cdot \deg(v)$. La pertinence de telles fonctions de poids apparaît ci-dessous :

Lemme 2.6 *Si $w : V \to \mathbf{Q}^+$ est une fonction de poids par degré, alors $w(V) \leqslant 2 \cdot \mathrm{OPT}$.*

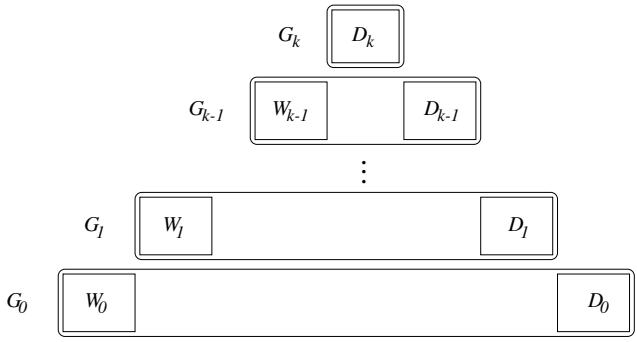
Preuve : Soient c la constante telle que $w(v) = c \cdot \deg(v)$, et C une couverture optimale de G. Puisque C couvre toutes les arêtes,

$$\sum_{v \in C} \deg(v) \geqslant |E|$$

Ainsi, $w(C) \geqslant c|E|$. Or, comme $\sum_{v \in V} \deg(v) = 2|E|$, $w(V) = 2c|E| \leqslant 2w(C)$. □

Nous définissons la *fonction de poids par degré maximale sous w* comme suit : après avoir éliminé tous les sommets de degré nul, calculer $c = \min\{w(v)/\deg(v)\}$; alors $t(v) = c \cdot \deg(v)$ est la fonction souhaitée. $w'(v) = w(v) - t(v)$ est la *fonction de poids résiduel*.

L'algorithme de décomposition de w en fonctions de poids par degré est le suivant. Posons $G_0 = G$. Éliminons tous les sommets de degré nul de G_0 ; notons D_0 cet ensemble. Calculons la fonction de poids par degré maximale sous w. Soit W_0 l'ensemble des sommets de poids résiduel nul ; ces sommets sont ajoutés à la couverture. Soit G_1 le graphe induit par $V \setminus (D_0 \cup W_0)$. Nous itérons le processus sur G_1 muni de la fonction de poids résiduel. L'algorithme est terminé lorsque tous les sommets sont de degré nul ; soit G_k ce graphe. La figure suivante illustre schématiquement le déroulement de l'algorithme.



Notons $t_0, ..., t_{k-1}$ les fonctions de poids par degré définies sur les graphes $G_0, ..., G_{k-1}$ respectivement. La couverture obtenue est $C = W_0 \cup \ldots \cup W_{k-1}$. Clairement, $V \setminus C = D_0 \cup \ldots \cup D_k$.

Théorème 2.7 *L'algorithme du mille-feuille est une* 2*-approximation pour le problème de la couverture par sommets munis de poids arbitraires.*

Preuve : Nous devons démontrer que C est une couverture par sommets de G et que $w(C) \leqslant 2 \cdot \mathrm{OPT}$. Supposons, par l'absurde, que C ne couvre pas G. Il existe alors $i \leqslant j$ et une arête uv telle que $u \in D_i$ et $v \in D_j$. Alors, l'arête uv est présente dans le graphe G_i, ce qui contredit que u soit de degré nul.

Soit C^* une couverture par sommets optimale. Considérons tout d'abord un sommet $v \in C$. Comme $v \in W_j$ pour un certain j, son poids peut s'écrire de la façon suivante :

$$w(v) = \sum_{i \leqslant j} t_i(v)$$

Considérons maintenant un sommet $v \in V \setminus C$. Alors $v \in D_j$ pour un certain j, et son poids est minoré par :

$$w(v) \geqslant \sum_{i < j} t_i(v)$$

Le fait important est que pour tout i, $C^* \cap G_i$ couvre la couche G_i, car G_i est un sous-graphe induit de G. Le lemme 2.6 implique donc : $t_i(C \cap G_i) \leqslant 2 \cdot t_i(C^* \cap G_i)$. Les réécritures des poids ci-dessus donnent alors :

$$w(C) = \sum_{i=0}^{k-1} t_i(C \cap G_i) \leqslant 2 \sum_{i=0}^{k-1} t_i(C^* \cap G_i) \leqslant 2 \cdot w(C^*)$$

□

Exemple 2.8 La famille des graphes bipartis complets $K_{n,n}$ munis d'une fonction de poids constante à 1 forme une famille d'instances critiques. L'algorithme du mille-feuille sélectionne les $2n$ sommets de $K_{n,n}$ dans la couverture, alors que la couverture optimale ne contient qu'une des deux moitiés de la bipartition. □

2.3 Application au problème du surfacteur minimum

Nous présentons l'algorithme suivant pour démontrer la généralité du problème de la couverture par ensembles. Nous donnerons au chapitre 7 une approximation à un facteur constant pour le problème du surfacteur minimum.

Commençons par présenter quelques applications de ce problème. On peut voir l'ADN humain comme un mot très long sur un alphabet à quatre lettres, que les scientifiques essayent de déchiffrer. Comme ce mot est très long, ils n'ont tout d'abord déchiffré que de courts morceaux de ce mot, qui se chevauchent les uns les autres. Bien entendu, les positions des morceaux dans le mot de l'ADN original sont inconnues. Une hypothèse est que le mot le plus court qui admet ces morceaux pour facteur[3], est une bonne approximation de l'ADN original.

Problème 2.9 (Surfacteur minimum)[4] Étant donné un alphabet fini Σ, et un ensemble de n mots, $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$, trouver un mot s de longueur minimum qui contienne une occurrence de chaque s_i. Sans perte de généralité, nous pouvons supposer qu'aucun des mots s_i n'est facteur d'un autre s_j, pour $j \neq i$.

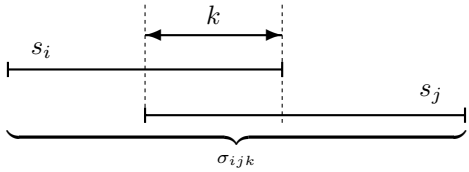
Ce problème est **NP**-difficile. Le premier algorithme auquel on pense pour trouver un surfacteur minimum est sans doute l'algorithme glouton suivant. Appelons *chevauchement*[5] d'un mot s sur un mot t le plus long suffixe de s qui est un préfixe de t. L'algorithme utilise un ensemble de mots T ; initialement $T = S$. À chaque étape, l'algorithme sélectionne deux mots de T de chevauchement maximum et les remplace par le surfacteur obtenu en les chevauchant au maximum. Au bout de $n-1$ étapes, T ne contient plus qu'un unique mot. Ce mot contient clairement une occurrence de chaque s_i. On conjecture que cet algorithme est une 2-approximation. Le facteur d'approximation est en tout cas minoré par 2. Considérons l'instance suivante $S = \{ab^k, b^kc, b^{k+1}\}$. Si l'algorithme glouton sélectionne les deux premiers mots à la première étape, il renvoie le mot ab^kcb^{k+1}. C'est asymptotiquement le double de la longueur du surfacteur minimum : $ab^{k+1}c$.

[3] On dit qu'un mot v est un *facteur* d'un mot u, s'il existe deux mots x et y tels que $u = xvy$, c'est-à-dire si u contient une occurrence de v. On dit alors que u est un *surfacteur* de v.

[4] *Shortest superstring*, en anglais.

[5] *Overlap*, en anglais.

Nous allons construire une $2H_n$-approximation en utilisant l'algorithme glouton pour la couverture par sommets. L'instance de la couverture par sommets, que nous noterons $\mathcal{S}$, est définie ainsi. Pour chaque $s_i, s_j \in S$ et $k > 0$, tels que le suffixe de longueur k de s_i est un préfixe de s_j, notons σ_{ijk} le mot obtenu en recouvrant les suffixe et préfixe de longueur k de s_i et s_j respectivement :



Notons M l'ensemble des mots σ_{ijk}, pour tous les choix valides de i, j, k. Pour chaque mot $\pi \in \Sigma^+$, notons $\text{set}(\pi) = \{s \in S : s \text{ est un facteur de } \pi\}$. S est l'ensemble univers de l'instance $\mathcal{S}$ de la couverture par sommets, et $\{\text{set}(\pi) : \pi \in S \cup M\}$ est la collection de sous-ensembles associée. Le coût de chaque $\text{set}(\pi)$ est la longueur de π, $|\pi|$.

Notons respectivement $\text{OPT}_{\mathcal{S}}$ et OPT le coût d'une solution optimale de $\mathcal{S}$ et la longueur d'un plus court surfacteur de S. Le lemme 2.11 démontre que $\text{OPT}_{\mathcal{S}}$ et OPT sont à un facteur 2 l'un de l'autre, et donc que l'on peut obtenir un algorithme d'approximation pour le problème du surfacteur minimum à partir d'un algorithme d'approximation pour la couverture par ensembles. Voici l'algorithme en détail :

Algorithme 2.10 (Surfacteur minimum via Couverture par ensembles)

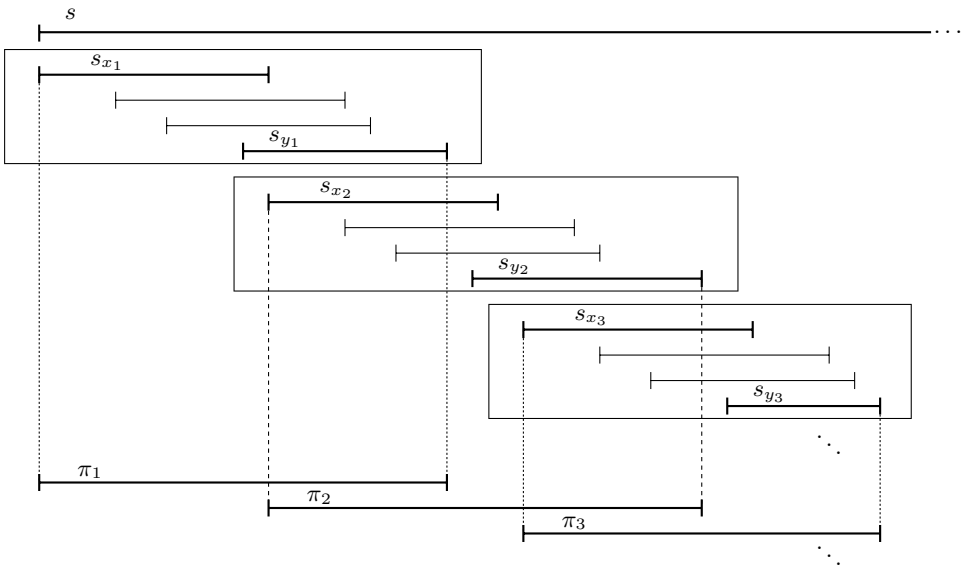
1. Utiliser l'algorithme glouton de couverture par ensembles sur l'instance $\mathcal{S}$. Soient $\text{set}(\pi_1)$, ..., $\text{set}(\pi_k)$ les ensembles sélectionnés dans cette couverture.
2. Concaténer les mots $\pi_1, \ldots, \pi_k$, dans n'importe quel ordre.
3. Renvoyer le mot $s = \pi_1 \pi_2 \ldots \pi_k$ obtenu.

Lemme 2.11 $\quad \text{OPT} \leqslant \text{OPT}_{\mathcal{S}} \leqslant 2 \cdot \text{OPT}$

Preuve : Soit $\{\text{set}(\pi'_i) | 1 \leqslant i \leqslant l\}$ une couverture optimale de $\mathcal{S}$, et s' un mot obtenu en concaténant les mots π'_i, $1 \leqslant i \leqslant l$, dans n'importe quel ordre. Par définition, $|s'| = \text{OPT}_{\mathcal{S}}$. Chaque mot de S est facteur d'un des π'_i, $1 \leqslant i \leqslant l$, et donc aussi facteur de s'. Par conséquent $\text{OPT}_{\mathcal{S}} = |s'| \geqslant \text{OPT}$.

Soit s^* un surfacteur minimum de $s_1, \ldots, s_n$, $|s^*| = \text{OPT}$. Pour démontrer la seconde inégalité, il suffit d'exhiber *une* couverture par ensembles de coût inférieur à $2 \cdot \text{OPT}$.

Étudions les positions relatives des premières occurrences (c'est-à-dire les plus à gauche) de chacun des mots $s_1, \ldots, s_n$ dans le mot s^*. Puisqu'aucun des $s_1, \ldots, s_n$, n'est facteur d'un autre, les positions de départ de ces n occurrences sont toutes distinctes. Pour la même raison, les positions des fins de ces occurrences sont toutes distinctes aussi et dans le même ordre. Réindexons les n mots dans l'ordre dans lequel ils apparaissent dans s^*.



Nous allons partitionner la liste ordonnée des mots $s_1, \ldots, s_n$ en l groupes comme suit. Chaque groupe est un sous-ensemble continu de mots de cette liste. Notons x_i et y_i les indices du premier et du dernier mot du i-ième groupe (il est possible que $x_i = y_i$). Nous posons $x_1 = 1$. y_1 est l'indice maximal d'un mot qui chevauche s_1 (il en existe au moins un, s_1). Tant que $y_i < n$, nous posons $x_{i+1} = y_i + 1$ et définissons y_{i+1} comme l'indice maximal d'un mot qui chevauche $s_{x_{i+1}}$. Nous obtenons ainsi t groupes avec $y_t = n$, avec $t \leqslant n$.

Pour chaque paire de mots (s_{x_i}, s_{y_i}), soit $k_i > 0$ la longueur du chevauchement de leurs premières occurrences dans s^* (ce qui peut être strictement inférieur à leur chevauchement maximum). Posons $\pi_i = \sigma_{x_i y_i k_i}$. Alors, $\{\mathrm{set}(\pi_i) | 1 \leqslant i \leqslant t\}$ est une couverture pour $\mathcal{S}$ de coût $\sum_i |\pi_i|$.

Le fait fondamental est qu'aucun π_i ne chevauche π_{i+2} dans s^*. Car, par construction, le premier mot de $(i+2)$-ième groupe ne chevauche pas le premier mot du $(i+1)$-ième, dont la fin est positionnée, dans s^*, après celle du dernier mot du i-ième groupe.

Ainsi, chaque symbole de s^* est couvert par au plus deux des π_i. Par conséquent, $\mathrm{OPT}_{\mathcal{S}} \leqslant \sum_i |\pi_i| \leq 2 \cdot \mathrm{OPT}$. □

La taille de l'ensemble univers de l'instance de couverture par ensembles $\mathcal{S}$ est n, le nombre de mots dans l'instance originale du surfacteur minimum. Ainsi, le lemme 2.11 et le théorème 2.4 donnent le théorème suivant.

Théorème 2.12 *L'algorithme 2.10 est une $2H_n$-approximation pour le problème du surfacteur minimum, où n est le nombre de mots de l'instance.*

2.4 Exercices

2.1 Étant donné un graphe non orienté $G = (V, E)$, le *problème de la coupe maximum* consiste à trouver une partition des sommets en deux ensembles S et $\overline{S}$ tels que le nombre d'arêtes entre eux soit maximum. Considérons l'algorithme glouton suivant. Soient v_1 et v_2 deux sommets arbitraires de G. Pour $A \subset V$, notons $d(v, A)$ le nombre d'arêtes ayant pour extrémité v et l'autre dans A.

Algorithme 2.13

1. Initialisation :
   $A \leftarrow \{v_1\}$
   $B \leftarrow \{v_2\}$
2. Pour chaque $v \in V - \{v_1, v_2\}$ faire :
   Si $d(v, A) \geqslant d(v, B)$ alors $B \leftarrow B \cup \{v\}$,
   sinon $A \leftarrow A \cup \{v\}$.
3. Renvoyer A et B.

Démontrez qu'il s'agit d'une 1/2-approximation et donnez une famille d'instances critiques. Quel majorant de OPT utilisez-vous ? Donnez des exemples de graphes pour lesquels ce majorant vaut $2 \cdot \mathrm{OPT}$.

Généralisez le problème et l'algorithme au cas où il existe une fonction de poids sur les sommets.

2.2 Considérons l'algorithme suivant pour le problème de la coupe maximum, fondé sur les techniques de *recherche locale*. Étant donné une partition de V en deux ensembles, chaque pas élémentaire de l'algorithme, appelé *échange*,[6] consiste à déplacer un sommet d'un côté à l'autre de la partition. L'algorithme suivant trouve une *solution localement optimale* au sens de l'échange, c'est-à-dire une solution qui ne peut pas être améliorée par un unique échange.

L'algorithme commence par une partition arbitraire de V. Tant qu'il existe un sommet dont l'échange augmente la taille de la coupe, l'algorithme échange

[6] *Flip*, en anglais.

ce sommet. (Remarquons que l'échange a lieu si le sommet a plus de voisins dans son côté de la partition que dans l'autre). L'algorithme termine lorsque plus aucun sommet ne vérifie la condition. Démontrez que cet algorithme termine en temps polynomial et que c'est une 1/2-approximation.

2.3 Considérons la généralisation suivante du problème de la coupe maximum.

Problème 2.14 (MAX k-CUT) Étant donné un graphe non orienté $G = (V, E)$ muni d'une fonction de coût positive sur les arêtes et un entier k, trouver une partition de V en k sous-ensembles $S_1, \ldots, S_k$ tels que le coût total des arêtes entre ces ensembles soit maximal.

Donnez un algorithme glouton pour ce problème qui soit une $(1 - \frac{1}{k})$-approximation. L'analyse de votre algorithme est-elle au plus près ?

2.4 Donnez un algorithme glouton pour le problème suivant qui soit une 1/4-approximation.

Problème 2.15 (Coupe maximum orientée)[7] Étant donné un graphe orienté $G = (V, E)$ muni d'une fonction de coût positive sur les arcs, trouver un sous-ensemble $S \subset V$ tel qu'il maximise le coût total des arcs qui sortent de S, c'est-à-dire $\sum_{u \in S, v \in \overline{S}} \text{coût}(u \to v)$.

2.5 (N. Vishnoi) Donnez une $\lceil \log_2 \Delta \rceil$-approximation pour le problème de la couverture par sommets (où Δ est le degré maximum d'un sommet de G) en vous fondant sur l'algorithme de l'exercice 2.1 et le fait que le problème de la couverture par sommets se résout en temps polynomial lorsque le graphe est biparti.
Indication : Soit H, le sous-graphe formé par les arêtes de la coupe maximale calculée par l'algorithme 2.13. H est biparti et pour tout sommet v, $\deg_H(v) \geqslant (1/2) \deg_G(v)$.

2.6 (Wigderson [266]) Considérons le problème suivant.

Problème 2.16 (Coloration de sommets)[8] Étant donné un graphe non orienté $G = (V, E)$, colorier avec un nombre minimal de couleurs ses sommets de sorte que les extrémités de chaque arête reçoivent des couleurs différentes.

1. Donnez un algorithme glouton qui colorie G avec $\Delta + 1$ couleurs, où Δ est le degré maximum d'un sommet de G.
2. Donnez un algorithme pour colorier avec $O(\sqrt{n})$ couleurs, un graphe coloriable avec trois couleurs.

   Indication : Pour tout sommet v, le graphe induit par ses voisins, $N(v)$, est biparti, donc coloriable de façon optimale. Si v est de degré $> \sqrt{n}$,

[7] *Maximum directed cut*, en anglais.
[8] *Vertex coloring*, en anglais.

coloriez $v \cup N(v)$ en utilisant trois couleurs distinctes. Continuez jusqu'à ce que tous les sommets soient de degré $\leqslant \sqrt{n}$. Puis, utilisez l'algorithme de la première partie.

2.7 Nous appellerons 2SC la restriction du problème de la couverture par ensembles aux instances où $f = 2$. Démontrez que 2SC est équivalent au problème de la couverture par sommets munis de poids arbitraires, et ce au travers de réductions isofacteurs.

2.8 Démontrez que l'algorithme 2.2 est une H_k-approximation, où k est la taille du plus grand ensemble de $\mathcal{S}$ (plus précisément du plus grand ensemble sélectionné dans une couverture optimale).

2.9 Proposez une H_n-approximation, sous la forme d'un algorithme glouton, pour le problème de la multicouverture par ensembles.[9] Ce problème est la généralisation de la couverture par ensembles, où chaque élément doit être couvert un nombre spécifié de fois, et où les ensembles peuvent être sélectionnés plusieurs fois. Considérez que le coût de la sélection de α copies d'un ensemble S_i est $\alpha \cdot \text{coût}(S_i)$.

2.10 En exhibant une famille d'instances critiques adéquate, démontrez que l'analyse de l'algorithme 2.2 ne peut pas être améliorée même pour le problème de la taille de la couverture par ensembles, c'est-à-dire lorsque les coûts des ensembles sont tous unitaires.
Indication : Simulez l'exécution de l'algorithme glouton sur une instance du problème de la couverture par sommets.

2.11 Étudions l'algorithme 2.17 ci-après pour le problème de la couverture par sommets. À chaque sommet v est associée une variable $t(v)$, initialisée au poids de v. $t(v)$ décroît au cours de l'algorithme. Lorsque $t(v)$ atteint zéro, v est sélectionné. $c(e)$ est le prix payé par l'arête e.
Démontrez que cet algorithme est une 2-approximation.
Indication : Démontrez que le prix total payé par les arêtes est un minorant de OPT et que le poids de la couverture C est au plus le double de ce prix.

[9] *Set multicover*, en anglais.

Algorithme 2.17

1. Initialisation :
   $C \leftarrow \varnothing$
   $\forall v \in V$, $t(v) \leftarrow w(v)$
   $\forall e \in E$, $c(e) \leftarrow 0$
2. Tant que C n'est pas une couverture par sommets faire :
   Choisir une arête non couverte, $\{u, v\}$. Poser $m = \min(t(u), t(v))$.
   $t(u) \leftarrow t(u) - m$
   $t(v) \leftarrow t(v) - m$
   $c(u, v) \leftarrow m$
   Inclure dans C tous les sommets tels que $t(v) = 0$.
3. Renvoyer C.

2.12 Considérons l'algorithme du mille-feuille pour trouver une couverture par sommets. Nous avons une 2-approximation pour d'autres fonctions de poids : les fonctions constantes — en utilisant tout simplement la 2-approximation pour le problème de la couverture par sommets de taille minimale. La méthode du mille-feuille fonctionne-t-elle toujours en utilisant des fonctions de poids constantes au lieu des fonctions de poids par degré ?

2.13 Donnez une f-approximation pour la couverture par ensembles, fondée sur la méthode du mille-feuille (où f est la fréquence maximale d'un élément). Exhibez une famille d'instances critiques pour votre algorithme.

2.14 Un *tournoi* est un graphe orienté $G = (V, E)$, tel que pour toute paire de sommets $(u, v) \in V^2$, une et une seule des deux arêtes (u, v) ou (v, u) est dans E. Un *coupe-cycles de sommets* (ou une *couverture par sommets des cycles*) de G est un ensemble de sommets de G tel que le sous-graphe induit par leur suppression est acyclique. Donnez une 3-approximation pour trouver une couverture par sommets des cycles d'un graphe orienté de taille minimum.
Indication : Démontrez qu'il suffit de « casser » tous les cycles de longueur 3. Utilisez la f-approximation pour la couverture par ensembles minimum.

2.15 (Hochbaum [133]) Considérons le problème suivant.

Problème 2.18 (Couverture Maximum)[10] Étant donné un univers U contenant n éléments munis de poids positifs, une collection de sous-ensembles $S_1, \ldots, S_l$ de U et un entier k, sélectionner k ensembles de façon à maximiser le poids total des éléments couverts.

[10] *Maximum coverage*, en anglais.

Démontrez que l'algorithme glouton trivial, qui sélectionne le meilleur ensemble à chaque étape jusqu'à ce que k ensembles soient choisis, est une approximation de facteur :

$$1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}$$

2.16 En vous appuyant sur le problème de la couverture par sommets, donnez des algorithmes d'approximation pour les variantes suivantes du surfacteur minimum (on note ici s^R le mot miroir de s) :

1. Trouvez un mot minimum contenant une occurrence de s_i et une de s_i^R pour tout mot $s_i \in S$.
   Indication : Prenez pour univers l'ensemble des s_i et s_i^R pour $1 \leqslant i \leqslant n$.
2. Trouvez un mot minimum contenant une occurrence de s_i ou une de s_i^R, pour tout mot $s_i \in S$.
   Indication : Définissez $\text{set}(\pi) = \{s \in S \mid s \text{ ou } s^R \text{ est un facteur de } \pi\}$. Choisissez bien les mots π.

2.5 Notes

L'algorithme 2.2 est dû à Johnson [158], Lovász [200], et Chvátal [51]. La preuve de l'inapproximabilité de la couverture par ensembles, démontrant que cet algorithme est essentiellement le meilleur possible, fut donnée par Feige [87], qui a amélioré le résultat de Lund et Yannakakis [207]. L'application au surfacteur minimum a été proposée par Li [195].

3 L'arbre de Steiner et le voyageur de commerce

Nous présentons dans ce chapitre des algorithmes d'approximation pour deux problèmes fondamentaux : l'arbre de Steiner métrique et le voyageur de commerce métrique. Les études des cas métriques de ces deux problèmes ont des justifications différentes. Pour l'arbre de Steiner, c'est le cœur du problème – le problème général se réduit au cas métrique. Le problème du voyageur de commerce, quant à lui, n'admet aucune approximation garantie sans cette restriction, à moins que $\mathbf{P} = \mathbf{NP}$. Nous présentons ces deux problèmes ensemble car les algorithmes et leurs analyses sont similaires.

3.1 L'arbre de Steiner métrique

Le problème de l'arbre de Steiner a été formalisé par Gauss dans une lettre (reproduite en couverture) qu'il écrivit à Schumacher. Aujourd'hui, ce problème occupe une place centrale en algorithmique d'approximation. Il a de très nombreuses applications, allant du réseau d'interconnexions de longueur minimale entre les différents éléments d'un circuit VLSI, à la construction d'arbres phylogéniques en bioinformatique. Ce problème et ses généralisations sont étudiés de façon approfondie dans ce livre (voir chapitres 22 et 23).

Problème 3.1 (Arbre de Steiner)[1] Étant donné un graphe non orienté $G = (V, E)$ muni d'une fonction de coût positive sur les arêtes, et dont les sommets sont étiquetés *requis* ou *Steiner*, trouver l'arbre coût minimum dans G qui contienne tous les sommets étiquetés requis et n'importe quel sous-ensemble des sommets étiquetés Steiner.

Nous allons commencer par démontrer que le cœur du problème est sa restriction au cas où la fonction de coût sur les arêtes vérifie l'*inégalité triangulaire*, c'est-à-dire le cas où G est un graphe complet et que pour tout triplet de sommets u, v, et w, $\text{coût}(uv) \leqslant \text{coût}(uw) + \text{coût}(vw)$. Nous appellerons cette restriction, le problème de l'*arbre de Steiner métrique*.

Théorème 3.2 *Il existe une réduction isofacteur du problème de l'arbre de Steiner à celui de l'arbre de Steiner métrique.*

[1] *Steiner tree*, en anglais.

Preuve : Transformons, en temps polynomial, toute instance I du problème l'arbre de Steiner sur un graphe $G = (V, E)$, en une instance I' du problème de l'arbre de Steiner métrique. Soit G' le graphe non orienté complet construit sur les sommets de V. Nous donnons pour coût à chaque arête uv de G celui du plus court chemin entre u et v dans G. G' s'appelle la *fermeture métrique*[2] de G. Les sommets de V sont étiquetés de la même façon dans I' que dans I.

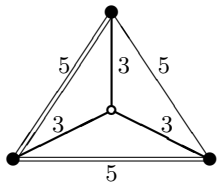
Remarquons que toute arête $uv \in E$ a un coût inférieur (ou égal) dans G' à celui dans G. Ainsi, le coût d'une solution optimale de I' est inférieur (ou égal) à celui d'une solution optimale de I : $\mathrm{OPT}' \leqslant \mathrm{OPT}$.

Démontrons maintenant comment obtenir, en temps polynomial, un arbre de Steiner T pour I à partir d'un arbre de Steiner T' de I', et dont le coût soit au plus celui de T'. Le coût de chaque arête uv de G' correspond au coût d'un chemin dans G. Remplaçons chaque arête de T' par son chemin correspondant dans G. Nous obtenons ainsi un sous-graphe de G. Tous les sommets étiquetés requis sont bien connectés dans ce sous-graphe. Cependant, ce sous-graphe peut contenir des cycles. Si tel est le cas, nous en ôtons des arêtes jusqu'à obtenir un arbre T. La réduction ainsi décrite est bien isofacteur, car $\frac{\text{coût}(T)}{\mathrm{OPT}} \leqslant \frac{\text{coût}(T')}{\mathrm{OPT}'}$ (et donc $\mathrm{OPT}' = \mathrm{OPT}$). □

Un corollaire du théorème 3.2 est que toute approximation garantie pour un facteur donné pour l'arbre de Steiner métrique produit la même garantie dans le cas général.

3.1.1 Approximation fondée sur l'arbre couvrant de poids minimum

Notons R, l'ensemble des sommets étiquetés requis. Un arbre couvrant de poids minimum (en abrégé, MST)[3] du sous-graphe induit par R est évidemment une solution réalisable du problème. Étant donné que le problème du MST est dans **P** et que celui de l'arbre de Steiner est **NP**-difficile, on ne peut espérer que le MST des sommets de R correspond toujours à un arbre de Steiner optimal ; l'exemple ci-dessous présente un MST (arêtes dédoublées) de coût strictement supérieur à l'optimal (en gras).

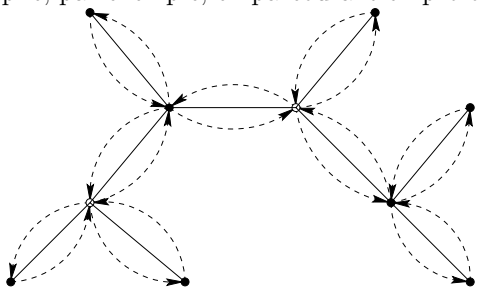


Cependant, un MST des sommets de R ne peut pas être beaucoup plus cher qu'un arbre de Steiner optimal :

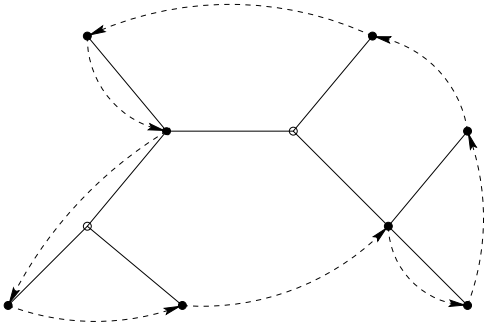
[2] *Metric closure*, en anglais.
[3] *Minimum spanning tree (MST)*, en anglais.

Théorème 3.3 *Le coût d'un MST du sous-graphe induit par les sommets de R est inférieur à $2 \cdot \mathrm{OPT}$.*

Preuve : Prenons un arbre de Steiner optimal. En dédoublant toutes ses arêtes, nous obtenons un graphe eulérien connectant tous les sommets de R, et éventuellement quelques sommets étiquetés Steiner. Construisons un cycle eulérien de ce graphe, par exemple, en parcourant en profondeur ses arêtes :



Le coût du cycle eulérien est $2 \cdot \mathrm{OPT}$. Construisons ensuite un cycle hamiltonien des sommets de R en parcourant le cycle eulérien et en court-circuitant[4] les sommets étiquetés Steiner ainsi que les sommets de R déjà visités :

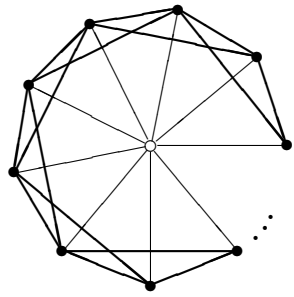


Grâce à l'inégalité triangulaire, ces sauts n'augmentent pas le coût du cycle. En éliminant une arête du cycle hamiltonien, nous obtenons un chemin passant par tous les sommets de R et de coût inférieur à $2 \cdot \mathrm{OPT}$. Ce chemin est un arbre couvrant de R. Un MST de R est donc nécessairement de coût inférieur à $2 \cdot \mathrm{OPT}$. □

Le théorème 3.3 décrit une 2-approximation directe pour le problème de l'arbre de Steiner métrique : calculer un MST des sommets étiquetés requis. De même que pour la couverture par ensembles, la « bonne » façon de voir cet algorithme est la théorie de la dualité en programmation linéaire. Nous verrons aux chapitres 22 et 23 que la dualité en programmation linéaire donne le minorant sur lequel cet algorithme est construit et aide à résoudre les généralisations de ce problème.

[4] *Short-cutting*, en anglais.

Exemple 3.4 Le graphe suivant est une instance critique. Il a n sommets requis et un sommet Steiner. Les arêtes entre le sommet Steiner et un sommet requis coûtent 1 et les arêtes entre deux sommets requis coûtent 2 (toutes les arêtes de coût 2 ne sont pas représentées ci-dessous). Dans ce graphe, tout MST de R coûte $2(n-1)$, alors que $\mathrm{OPT} = n$.



□

3.2 Le voyageur de commerce métrique

Le problème suivant a été très étudié en optimisation combinatoire.

Problème 3.5 (Voyageur de commerce – TSP)[5] Étant donné un graphe complet muni d'une fonction de coût positive sur les arêtes, trouver un cycle de coût minimum passant par chaque sommet exactement une fois.

Dans toute sa généralité, on ne peut garantir aucune approximation pour TSP, à moins que $\mathbf{P} = \mathbf{NP}$.

Théorème 3.6 *Quel que soit $\alpha(n)$ calculable en temps polynomial, il n'existe pas de $\alpha(n)$-approximation pour TSP, à moins que* $\mathbf{P} = \mathbf{NP}$.

Preuve : Par l'absurde, soit $\mathcal{A}$ une $\alpha(n)$-approximation (en temps polynomial) pour la version générale du TSP. Nous allons démontrer que $\mathcal{A}$ permet de résoudre le problème **NP**-difficile du cycle hamiltonien en temps polynomial, et donc que $\mathbf{P} = \mathbf{NP}$.

L'idée est de réduire le problème du cycle hamiltonien au TSP en transformant un graphe G à n sommets en un graphe complet G' à n sommets muni d'une fonction de poids sur les arêtes telle que :

- si G admet un cycle hamiltonien, alors le coût optimal du TSP dans G' est n, et
- si G n'admet pas de cycle hamiltonien, alors le coût optimal du TSP dans G' est $> \alpha(n) \cdot n$.

[5] *Traveling salesman problem (TSP)*, en anglais.

Remarquons que, sur le graphe G', l'algorithme $\mathcal{A}$ calcule une solution de coût $\leqslant \alpha(n) \cdot n$ dans le premier cas, et $> \alpha(n) \cdot n$ dans le second. Ainsi, nous pouvons l'utiliser pour décider si G admet ou non un cycle hamiltonien.

La réduction est simple. Nous attribuons, dans G', un coût de 1 aux arêtes également présentes dans G, et un coût de $\alpha(n) \cdot n$ aux autres. Ainsi, si G admet un cycle hamiltonien, le cycle correspondant dans G' coûte n. Et si G n'admet pas de cycle hamiltonien, tout cycle de G' passe nécessairement pas une arête de coût $\alpha(n) \cdot n$, et donc coûte strictement plus que $\alpha(n) \cdot n$. □

Remarquons que pour obtenir un résultat d'inapproximabilité aussi fort, nous avons dû assigner aux arêtes des coûts qui ne respectent pas l'inégalité triangulaire. Lorsque l'on se restreint aux graphes munis d'une fonction de coût métrique (qui respecte l'inégalité triangulaire), c'est-à-dire à TSP métrique, le problème reste **NP**-difficile, mais il n'est plus impossible d'en approcher la solution optimale.

3.2.1 Une 2-approximation facile

Commençons par présenter une 2-approximation facile. Le minorant que nous utiliserons est le coût d'un MST de G. C'est bien un minorant car on obtient un arbre couvrant de G en ôtant une arête d'une solution optimale du TSP.

Algorithme 3.7 (TSP métrique – 2-approximation)

1. Calculer un MST, T, de G.
2. Dupliquer toutes les arêtes du MST pour obtenir un graphe eulérien.
3. Calculer un cycle eulérien $\mathcal{T}$ de ce graphe.
4. Renvoyer le cycle $\mathcal{C}$ qui passe par tous les sommets de G dans l'ordre de leur première occurrence dans $\mathcal{T}$.

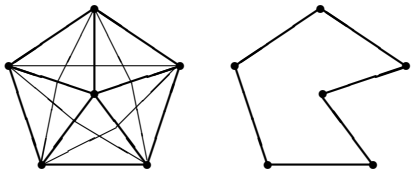
Remarquons que la quatrième étape est la même que les « court-circuitages » de sommets dans la preuve du théorème 3.3.

Théorème 3.8 *L'algorithme 3.7 est une* 2*-approximation pour le TSP métrique.*

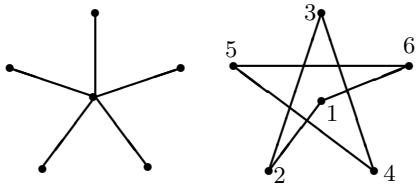
Preuve : Nous l'avons vu ci-dessus, $\text{coût}(T) \leqslant \text{OPT}$. Puisque $\mathcal{T}$ contient chaque arête de T en double, $\text{coût}(\mathcal{T}) = 2 \cdot \text{coût}(T)$. Par l'inégalité triangulaire, après la quatrième étape, $\text{coût}(\mathcal{C}) \leqslant \text{coût}(\mathcal{T})$. Ces deux inégalités permettent de conclure que $\text{coût}(\mathcal{C}) \leqslant 2 \cdot \text{OPT}$. □

Exemple 3.9 Voici une instance critique pour cet algorithme. C'est un graphe complet à n sommets dont les arêtes coûtent 1 ou 2. La figure ci-dessous représente le graphe pour $n = 6$ (les arêtes de coût 1 sont dessinées

en gras, les autres coûtent 2). Pour un n quelconque, ce graphe a $2n-2$ arêtes de coût 1, formant l'union d'une étoile et d'un cycle de longueur $n-1$; les autres arêtes coûtent 2. Le TSP optimal coûte n :



Supposons que le MST trouvé par l'algorithme soit l'étoile couvrante constituée d'arêtes de coût unitaire. Supposons également que le cycle eulérien calculé à la troisième étape passe par les sommets dans l'ordre ci-dessous (pour le cas $n = 6$) :



Alors, le cycle obtenu après la quatrième étape contient $n-2$ arêtes de coût 2 pour un coût total de $2n-2$. C'est asymptotiquement le double du cycle TSP optimal. □

3.2.2 Amélioration du facteur d'approximation à 3/2

L'algorithme 3.7 commence par calculer un cycle eulérien de faible coût des sommets de G, puis court-circuite ce cycle pour trouver un cycle de voyageur de commerce. Peut-on trouver un cycle eulérien plus économique, de coût inférieur au double du coût du MST ? Rappelons qu'un graphe admet un cycle eulérien si et seulement si tous ses sommets sont de degré pair. Ainsi, nous ne devons donc nous occuper que des sommets de degré impair du MST. Notons V' l'ensemble de ces sommets. Il n'existe qu'un nombre pair de sommets de degré impair car la somme des degrés des sommets d'un graphe est pair (c'est le double du nombre d'arêtes). Ainsi $|V'|$ est pair. Si nous ajoutons au MST un couplage parfait de coût minimum sur V', tous les sommets auront degré pair, et nous aurons notre graphe eulérien. Cette amélioration conduit à 3/2-approximation.

Algorithme 3.10 (TSP métrique – 3/2-approximation)

1. Calculer un MST de G, noté T.
2. Calculer un couplage parfait M de coût minimum du sous-graphe induit par les sommets de degré impair de T. Ajouter M à T pour obtenir un graphe eulérien.
3. Calculer un cycle eulérien $\mathcal{T}$ de ce graphe.
4. Renvoyer le cycle $\mathcal{C}$ qui passe par les sommets de G dans l'ordre de leur première occurrence dans $\mathcal{T}$.

Remarquons que l'évaluation des performances de cet algorithme repose sur un autre minorant de OPT.

Lemme 3.11 *Soient $V' \subseteq V$, avec $|V'|$ pair, et M un couplage parfait de coût minimum de V'. Alors* $\text{coût}(M) \leqslant \text{OPT}/2$.

Preuve : Soit γ le cycle TSP optimal de G. Soit γ' le cycle de V' obtenu à partir de γ en court-circuitant les sommets de $V \setminus V'$. Par l'inégalité triangulaire, $\text{coût}(\gamma') \leqslant \text{coût}(\gamma)$. Comme $|V'|$ est pair, τ' est l'union de deux couplages parfaits de V'. Par conséquent, le moins cher de ces deux couplages a un coût $\leqslant \text{coût}(\gamma')/2 \leqslant \text{OPT}/2$. Le couplage de coût minimum coûte donc moins que OPT/2. □

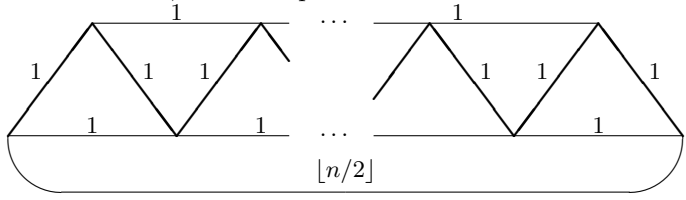
Théorème 3.12 *L'algorithme 3.10 est une* 3/2*-approximation pour le TSP métrique.*

Preuve : Les deux minorants de OPT impliquent que le coût du cycle eulérien est majoré par :

$$\text{coût}(\mathcal{T}) \leqslant \text{coût}(T) + \text{coût}(M) \leqslant \text{OPT} + \frac{1}{2}\,\text{OPT} = \frac{3}{2}\,\text{OPT}$$

L'inégalité triangulaire permet de conclure car $\text{coût}(\mathcal{C}) \leqslant \text{coût}(\mathcal{T})$. □

Exemple 3.13 Le graphe suivant est une instance critique pour cet algorithme. Il a n sommets, avec n impair :



Le MST trouvé à la première étape est représenté par des arêtes en gras. Ce MST n'a que deux sommets de degré impair. En ajoutant l'arête qui les joint,

on obtient un parcours pour le voyageur de commerce de coût $(n-1)+\lfloor n/2 \rfloor$. Or, le cycle optimal ne coûte que n. □

La recherche d'une meilleure approximation pour le TSP métrique est actuellement l'un des problèmes ouverts les plus importants du domaine. De nombreux chercheurs conjecturent qu'une approximation de facteur 4/3 est sans doute possible.

3.3 Exercices

3.1 La difficulté du problème de l'arbre de Steiner est de déterminer quel est le sous-ensemble optimal de sommets étiquetés Steiner à inclure dans l'arbre. Démontrez que si cet ensemble est donné, alors l'arbre de Steiner optimal se calcule en temps polynomial.
Indication : Recherchez un MST de l'union de cet ensemble et des sommets étiquetés requis.

3.2 Considérons un graphe $G = (V, E)$ muni d'une fonction de coût positive sur les arêtes, et deux sous-ensembles disjoints de V : $\mathcal{E}$, les *expéditeurs* et $\mathcal{D}$, les *destinataires*. Le problème est de trouver un sous-graphe de G de coût minimal tel que tout destinataire est relié par un chemin à l'un des expéditeurs (n'importe lequel). Séparons les instances en deux catégories : $\mathcal{E} \cup \mathcal{D} = V$ et $\mathcal{E} \cup \mathcal{D} \neq V$. Démontrez que ces deux cas sont respectivement dans **P** et **NP**-difficile. Proposez une 2-approximation pour le second cas.
Indication : Ajoutez un sommet supplémentaire relié à chaque expéditeur par une arête de coût nul. Étiquetez requis le nouveau sommet ainsi que tous les destinataires, puis Steiner les autres. Recherchez un arbre de Steiner de coût minimum.

3.3 Donnez une réduction isofacteur du problème de la couverture par ensembles au problème suivant — démontrant ainsi qu'il est improbable de trouver une approximation de facteur meilleur que $O(\log n)$.

Problème 3.14 (Arborescence de Steiner)[6] Soit $G = (V, E)$ un graphe orienté muni d'une fonction de coût positive sur les arêtes. Les sommets sont tous étiquetés *requis* ou *Steiner*. Un des sommets requis, r, est distingué. Le problème consiste à trouver une arborescence enracinée en r de coût minimal, qui contient tous les sommets requis et un sous-ensemble quelconque des sommets Steiner.
Indication : Construisez un graphe à trois couches : la première contient un sommet requis pour chaque élément de l'instance de la couverture par sommets ; la seconde contient un sommet Steiner pour chacun des sous-ensembles ; et la troisième contient r.

[6] *Directed Steiner tree*, en anglais.

3.4 (Hoogeveen [138]) Étudions des variantes du TSP métrique dans lesquelles il s'agit de trouver un chemin simple passant par tous les sommets du graphe. Trois problèmes différents se posent suivant que l'on fixe aucune, une ou deux des extrémités du chemin. Proposez des algorithmes ayant les garanties suivantes pour chacun des cas :

– si zéro ou une des extrémités est fixée, donnez une 3/2-approximation ;
– si les deux extrémités sont fixées, donnez une 5/3-approximation.

Indication : Inspirez-vous de l'algorithme 3.10.

3.5 (Papadimitriou et Yannakakis [228]) Soit G un graphe non orienté complet dont toutes les arêtes sont de longueur 1 ou 2 (G vérifie alors clairement l'inégalité triangulaire). Proposez une 4/3-approximation pour le TSP sur cette classe de graphes.
Indication : Commencez par trouver un 2-facteur minimal de G. Un 2-*facteur*[7] est un sous-ensemble S d'arêtes tel que chaque sommet est incident à exactement deux arêtes de S.

3.6 (Frieze, Galbiati, and Maffioli [96]) Donnez une $O(\log n)$-approximation pour le problème suivant.

Problème 3.15 (TSP asymétrique)[8] Étant donné un graphe orienté $G = (V, E)$ muni d'une fonction de coût sur les arcs satisfaisant l'*inégalité triangulaire orientée*, c'est-à-dire telle que pour tout triplet de sommets u, v, et w, $\text{coût}(u \to v) \leqslant \text{coût}(u \to w) + \text{coût}(w \to v)$. Le problème consiste à trouver un cycle de coût minimal qui passe par chacun des sommets exactement une fois.
Indication : Utilisez le fait qu'on peut calculer en temps polynomial une couverture par cycles de coût minimal (c'est-à-dire des cycles disjoints qui couvrent tous les sommets). Réduisez les cycles à un point et itérez.

3.7 Soient $G = (V, E)$ un graphe muni d'une fonction de coût sur les arêtes vérifiant l'inégalité triangulaire, et $V' \subset V$ un sous-ensemble de taille paire. Prouvez ou infirmez le fait suivant : « le coût minimum d'un couplage parfait sur V' est majoré par le coût minimum d'un couplage parfait V ».

3.8 Étant donné n points dans $\mathbf{R}^2$, nous appelons arbre de Steiner optimal, un arbre de longueur totale minimale qui atteint ces n points en passant pour cela par n'importe quels autres points du plan. Démontrez que le degré de chacun des points additionnels vaut 3, et que leurs arêtes incidentes forment des angles à 120°.

3.9 (Rao, Sadayappan, Hwang, and Shor [239]) Cet exercice construit une 2-approximation pour le problème suivant.

[7] 2-*Matching*, en anglais.
[8] *Asymetric TSP*, en anglais.

Problème 3.16 (Arborescence de Steiner rectilinéaire)[9] Considérons n points $p_1, \ldots, p_n$ du quadrant positif du plan $\mathbf{R}^2$. On dit qu'un chemin de l'origine à p_i est *monotone* s'il est composé de segments parallèles aux axes x et y dans le sens positif (de manière informelle, allant vers l'est ou vers le nord). Le problème est de trouver un arbre de longueur minimale constitué de chemins monotones partant des n points ; un tel arbre est appelé une *arborescence de Steiner rectilinéaire*.

Pour tout point p, nous notons (x_p, y_p) ses coordonnées et $|p|_1 = |x_p| + |y_p|$. Nous dirons qu'un point p *domine* un point q si $x_p \leqslant x_q$ et $y_p \leqslant y_q$. Étant donné deux ensembles de points A et B, nous dirons que A *domine* B si pour tout point $b \in B$, il existe un point $a \in A$ qui domine b. Étant donné deux points p et q, le *dominateur* de p et q est le point $\mathrm{dom}(p, q) = (x, y)$, avec $x = \min(x_p, x_q)$ et $y = \min(y_p, y_q)$. Lorsque p domine q, on note $\mathrm{segments}(p, q)$ un chemin monotone arbitraire de p à q. Voici l'algorithme.

Algorithme 3.17 (Arborescence de Steiner rectilinéaire)

1. $T \leftarrow \varnothing$.
2. $P \leftarrow \{p_1, \ldots, p_n\} \cup \{(0, 0)\}$.
3. Tant que $|P| > 1$ faire :
   Trouver des points $p, q \in P$, $p \neq q$, qui maximisent $|\mathrm{dom}(p, q)|_1$.
   $P \leftarrow (P - \{p, q\}) \cup \{\mathrm{dom}(p, q)\}$.
   $T \leftarrow T \cup \mathrm{segments}(\mathrm{dom}(p, q), p) \cup \mathrm{segments}(\mathrm{dom}(p, q), q)$.
4. Renvoyer T.

Pour tout $z \geqslant 0$, notons ℓ_z la droite d'équation $x + y = z$. Pour toute arborescence de Steiner T, soit $T(z) = |T \cap \ell_z|$, le nombre de points à l'intersection de T et ℓ_z. Démontrez que la longueur de T est :

$$\int_{z=0}^{\infty} T(z)\, \mathrm{d}z$$

Pour tout $z \geqslant 0$, posons $P_z = \{p \in P : |p|_1 > z\}$, et

$$N(z) = \min\{|C| \ : \ C \subset \ell_z \text{ et } C \text{ domine } P_z\}.$$

[9] *Rectilinear Steiner arborescence*, en anglais.

Démontrez que

$$\int_{z=0}^{\infty} N(z)\,\mathrm{d}z$$

est un minorant de OPT.

En conclure que l'algorithme 3.17 est une 2-approximation.

3.10 (I. Măndoiu) Cet exercice construit une 9-approximation pour le problème suivant, dont une application est le routage d'horloge VLSI.

Problème 3.18 (Arbre rectilinéaire isochrone) Étant donné un ensemble S de points du plan rectilinéaire (c'est-à-dire muni de la norme ℓ_1), trouver un *arbre isochrone*[10] (ZST) de S, de longueur minimum, c'est-à-dire une arborescence T du plan rectilinéaire dont les feuilles sont les points de S et dont tous les chemins de la racine à une feuille ont la même longueur. Nous entendons par *longueur d'un chemin*, la somme des longueurs de ses arêtes.

1. Soient T un arbre isochrone et R' la longueur commune des chemins de la racine aux feuilles. Pour tout $r \geqslant 0$, notons $T(r)$ le nombre de points de T qui sont à distance $R' - r$ de la racine (dans l'arbre). Démontrez que la longueur de T est :

$$\int_0^{R'} T(r)\mathrm{d}r.$$

2. On appelle boule ℓ_1 fermée de rayon r centrée en p l'ensemble des points dont la distance ℓ_1 de p est $\leqslant r$. Soit R le rayon de la plus petite boule ℓ_1 fermée qui contient tous les points de S. Pour tout $r \geqslant 0$, notons $N(r)$ le nombre minimal de boules ℓ_1 fermées de rayon r nécessaires pour couvrir tous les points de S. Démontrez que

$$\int_0^{R} N(r)\mathrm{d}r$$

est un minorant de la longueur d'un ZST optimum.

3. Le principe de l'algorithme est le suivant. Calculer R et trouver une boule ℓ_1 fermée de rayon R qui couvre les points de S. Le centre de cette boule sera la racine du ZST. Cette boule est découpée en quatre boules, nommées ses *quadrants*, de rayon $R/2$ chacune. La racine est connectée aux centres de chacune de ces boules par une arête de longueur $R/2$. Chacune de ces boules est ensuite découpée en quatre nouvelles boules de rayon $R/4$, ainsi de suite.

[10] *Zero-skew tree (ZST)*, en anglais.

L'algorithme construit le ZST récursivement en commençant par la boule de rayon R. Le centre de la boule courante est connecté aux centres de chacun de ses quadrants qui couvre (au moins) un point de S. L'algorithme itère récursivement sur chacun des quadrants. Si la boule courante couvre exactement un point de S, la récursion s'arête et la boule n'est pas découpée. Soient r' le rayon de cette boule, c son centre et p le point de S qu'elle couvre. Clairement, la distance ℓ_1 de c à p est $\leqslant r'$. On connecte alors c à p par un chemin rectilinéaire de longueur r' exactement.

Démontrez que l'arbre obtenu est bien isochrone, puis que $T(r) \leqslant 9N(r)$, pour tout $0 \leqslant r \leqslant R$. En déduire que cet algorithme est une 9-approximation.

3.4 Notes

Le problème de l'arbre de Steiner a ses origines dans un problème posé par Fermat. Il fut formalisé le 21 mars 1836 par Gauss dans une lettre adressée à son étudiant Schumacher. Des extraits de cette lettre sont reproduits sur la couverture de ce livre. Courant et Robbins [58] popularisèrent ce problème sous le nom de Steiner, un géomètre célèbre du dix-neuvième siècle. Nous renvoyons à Hwang, Richards, et Winter [141] et Schreiber [245] pour la fascinante histoire de ce problème.

La 2-approximation du problème de l'arbre de Steiner fut découverte indépendamment par Choukhmane [47], Iwainsky, Canuto, Taraszow et Villa [144], Kou, Markowsky et Berman [185], et Plesník [230]. La 3/2-approximation pour le TSP métrique est due à Christofides [48], et le théorème 3.6 à Sahni et Gonzalez [241]. Le minorant de l'exercice 3.10 fut proposé par Charikar, Kleinberg, Kumar, Rajagopalan, Sahai et Tomkins [43]. La meilleure approximation connue pour le problème l'arbre rectilinéaire isochrone, une 3-approximation, est due à Zelikovsky et Măndoiu [273].

Étant donné n points du plan euclidien, l'arbre couvrant de poids minimum est à un facteur $2/\sqrt{3}$ du coût optimal de l'arbre de Steiner (qui est autorisé à utiliser n'importe quel point du plan comme point Steiner). Ceci fut démontré par Du et Hwang [68], fermant ainsi une conjecture de Gilbert et Pollak [107].

4 Coupe multiséparatrice et coupe en k morceaux

La théorie des coupes joue un rôle central en algorithmique exacte. Il en va de même pour les algorithmes d'approximation. Nous présentons dans ce chapitre des algorithmes d'approximation pour des généralisations naturelles du problème de la coupe minimale qui sont toutes **NP**-difficiles.

Soit $G = (V, E)$ un graphe connexe non orienté muni d'une fonction de poids sur les arêtes $w : E \to \mathbf{R}^+$. Une *coupe*[1] de G est définie par une partition de V en deux ensembles V' et $V \setminus V'$, et consiste en l'ensemble des arêtes qui ont une extrémité de chaque côté de la partition. Clairement, le retrait d'une coupe non triviale de G déconnecte G. Considérons deux *terminaux*[2] $s, t \in V$. Une coupe définie par une partition de V qui sépare s et t, s'appelle une *coupe*[3] *entre* s *et* t. Une coupe de poids minimum ou une coupe entre s et t de poids minimum se calculent efficacement en utilisant un algorithme de flot maximum. Étudions les généralisations suivantes de ces deux notions :

Problème 4.1 (Coupe multiséparatrice)[4] Étant donné un ensemble de terminaux $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, une *coupe multiséparatrice* est un ensemble d'arêtes dont le retrait du graphe déconnecte les terminaux entre eux. Trouver une coupe multiséparatrice de poids minimum.

Problème 4.2 (Coupe en k morceaux de poids minimum)[5] Une *coupe en* k *morceaux* (ou k-coupe) d'un graphe est un ensemble d'arêtes dont le retrait déconnecte le graphe en k composantes connexes. Trouver une coupe en k morceaux de poids minimum.

Trouver une coupe multiséparatrice de poids minimum est un problème **NP**-difficile pour tout $k \geqslant 3$. Remarquons que pour $k = 2$, c'est précisément le problème de la coupe minimum entre s et t. On peut trouver une coupe en k morceaux en temps polynomial pour tout k fixé ; le problème est en revanche **NP**-difficile lorsque k fait partie des entrées. Dans ce chapitre, nous donnons des $(2 - 2/k)$-approximations pour ces deux problèmes. Au chapitre 19, nous

[1] *A cut*, en anglais.
[2] *The terminals*, en anglais.
[3] *A s–t cut*, en anglais.
[4] *Multiway cut*, en anglais.
[5] *k-cut*, en anglais.

améliorerons la garantie de performance à 3/2 pour le problème de la coupe multiséparatrice.

4.1 Le problème de la coupe multiséparatrice

Nous appellerons *coupe séparatrice*[6] pour s_i, un ensemble d'arêtes dont le retrait déconnecte s_i des autres terminaux.

Algorithme 4.3 (Coupe k-séparatrice)

1. Pour chaque $i = 1, \ldots, k$, calculer une coupe séparatrice C_i pour s_i de poids minimum.
2. Éliminer la plus lourde de ces coupes, puis renvoyer l'union C des coupes restantes.

Pour chaque $i = 1, \ldots, k$, la première étape fusionne les terminaux de $S \setminus \{s_i\}$ en un nœud unique, et calcule une coupe de poids minimum séparant ce nouveau nœud de s_i ; ceci ne requiert qu'un calcul de flot maximum. Clairement, le retrait de C déconnecte toutes les paires de terminaux, et forme donc une coupe multiséparatrice.

Théorème 4.4 *L'algorithme 4.3 est une* $(2 - 2/k)$*-approximation.*

Preuve : Soit A une coupe multiséparatrice optimale de G. A est l'union des k coupes suivantes. Le retrait des arêtes de A déconnecte G en k composantes connexes, contenant chacune un terminal (puisque A est de poids minimum, exactement k composantes connexes sont créées, en l'absence d'arêtes de poids nul). Notons A_i la coupe séparant la composante connexe contenant s_i du reste du graphe. On a : $A = \bigcup_{i=1}^{k} A_i$.

Puisque chaque arête de A est incidente à deux de ces composantes, chaque arête est incluse dans exactement deux coupes A_i. Ainsi,

$$\sum_{i=1}^{k} w(A_i) = 2w(A).$$

Chaque A_i est clairement une coupe séparatrice pour s_i. Puisque C_i est une coupe séparatrice pour s_i de poids minimum, $w(C_i) \leqslant w(A_i)$. Remarquons qu'on obtient donc une 2-approximation en prenant l'union des k coupes C_i. Or, puisque C est obtenue en éliminant la coupe la plus lourde des C_i,

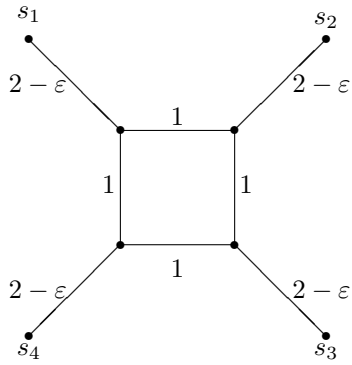
$$w(C) \leqslant \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(C_i) \leqslant \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(A_i) = 2\left(1 - \frac{1}{k}\right) w(A).$$

[6] *Isolating cut*, en anglais.

□

Une fois de plus, l'algorithme 4.3 ne repose pas directement sur un minorant. L'exercice 19.2 propose un algorithme avec la même garantie de performance, utilisant un minorant obtenu par relaxation d'un programme linéaire. L'utilisation d'une relaxation d'un programme linéaire est fructueuse pour ce problème. Nous verrons, section 19.1, comment construire un algorithme avec une meilleure garantie, en utilisant une autre relaxation linéaire.

Exemple 4.5 Une instance critique pour cet algorithme est obtenue en considérant le graphe à $2k$ sommets, formé d'un cycle de longueur k et de k terminaux distincts, reliés chacun à un sommet du cycle. Les arêtes du cycle sont toutes de poids 1, et le poids de chaque arête reliant un terminal au cycle est $2-\varepsilon$ pour $\varepsilon > 0$ suffisamment petit. Voici le graphe pour $k = 4$:



Pour chaque terminal s_i, la coupe séparatrice de poids minimum est l'unique arête incidente à s_i. Ainsi, la coupe C renvoyée par l'algorithme a pour poids $(k-1)(2-\varepsilon)$. Or, la coupe multiséparatrice optimale est donnée par les arêtes du cycle, et son poids est k. □

4.2 Coupe en k morceaux de poids minimum

Voici un algorithme naturel pour trouver une coupe en k morceaux. Commençant avec G, calculer une coupe minimale de chacune de ses composantes connexes, puis sélectionner et ôter de G la coupe la plus légère ; itérer jusqu'à ce qu'il y ait k composantes connexes. Cet algorithme est une $(2-2/k)$-approximation, mais la preuve en est cependant assez élaborée. Nous préférons présenter ici un algorithme plus simple, offrant la même garantie, fondé sur la *représentation arborescente de Gomory-Hu des coupes de poids minimum*.

Contrairement aux coupes dans les graphes orientés, les coupes minimum d'un graphe non orienté, de même que les coupes sous-optimales, ont des propriétés structurelles intéressantes (l'algorithme de la section 28.2 repose

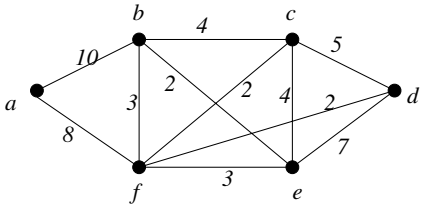
sur certaines de ces propriétés). L'existence d'arbres de Gomory-Hu, que nous allons définir dans un instant, est l'une des conséquences remarquables de ces propriétés.

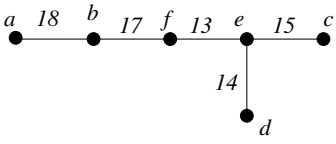
Soit T un arbre construit sur les sommets V — il n'est pas nécessaire que les arêtes de T soient dans E. Soit e une arête de T. Son retrait de T engendre deux composantes connexes. Notons S et $\overline{S}$ ces deux ensembles de sommets. La coupe définie dans le graphe G par la partition $(S, \overline{S})$ est la *coupe associée à e par T dans G*. Considérons une fonction de poids w' sur les arêtes de T. L'arbre pondéré T sera dit *de Gomory-Hu* pour G si

1. pour toute paire de sommets $u, v \in V$, le poids minimum d'une coupe entre u et v dans G est le même que dans T, et
2. pour toute arête $e \in T$, $w'(e)$ est le poids de sa coupe associée dans G.

Un arbre de Gomory-Hu encode donc succinctement des coupes de poids minimum séparant chaque paire de sommets de G. En effet, une coupe de poids minimum entre u et v dans T est donnée par la coupe associée dans G à une arête e de poids minimum sur l'unique chemin entre u et v dans T. Les propriétés ci-dessus garantissent que la coupe associée est bien une coupe de poids minimum entre u et v, ce poids étant $w'(e)$. Ainsi, nous n'avons besoin que de $n-1$ coupes, encodées par les arêtes d'un arbre de Gomory-Hu, pour trouver une coupe de poids minimum séparant chacune des $\binom{n}{2}$ paires de sommets de G.

La figure suivante présente un graphe pondéré et son arbre de Gomory-Hu associé. L'exercice 4.6 montre comment construire un arbre de Gomory-Hu pour un graphe non orienté, en utilisant seulement $n-1$ calculs de flot maximum.





Le lemme suivant nous sera utile par la suite.

Lemme 4.6 *Considérons l arêtes de T. Notons S l'union des coupes de G associées à ces l arêtes. Alors, le retrait de S dans le graphe G déconnecte G en au moins $l+1$ composantes connexes.*

Preuve : Le retrait de l arêtes dans T déconnecte T en exactement $l+1$ composantes connexes ; notons $V_1, V_2, \ldots, V_{l+1}$ les ensembles de sommets associés. Par définition, le retrait de S dans G déconnecte chaque sommet de V_i de ceux de V_j, pour $i \neq j$. Ainsi, le graphe résultant a au moins $l+1$ composantes connexes. □

Le lemme 4.6 implique donc que l'union de $k-1$ coupes choisies dans T forme une coupe en k morceaux de G. Voici donc l'algorithme.

Algorithme 4.7 (Coupe en k morceaux de poids minimum)

1. Calculer un arbre de Gomory-Hu T de G.
2. Renvoyer l'union C des $k-1$ coupes les plus légères parmi les $n-1$ coupes associées aux arêtes de T dans G.

Le lemme 4.6 garantit que le retrait de C dans G crée au moins k composantes connexes. Si l'on crée plus de k composantes connexes, il suffit de supprimer des arêtes de C jusqu'à ce qu'il y ait exactement k composantes connexes.

Théorème 4.8 *L'algorithme 4.7 est une $2-2/k$-approximation.*

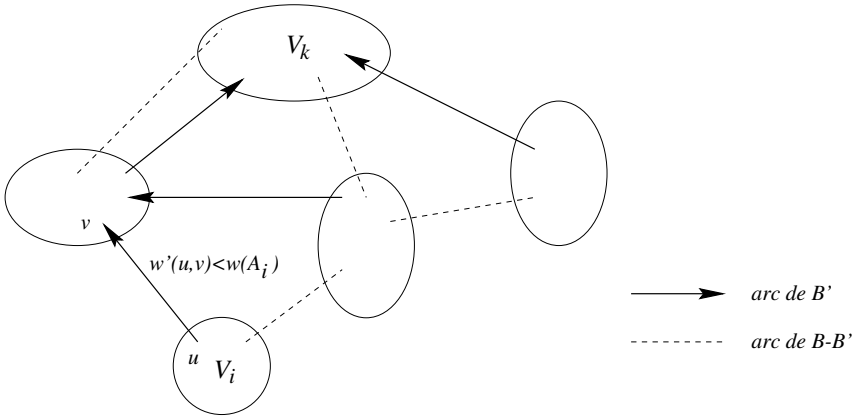
Preuve : Soit A une coupe optimale en k morceaux de G. À l'instar du théorème 4.4, voyons A comme l'union de k coupes. Soient $V_1, V_2, \ldots, V_k$, les k composantes connexes engendrées par le retrait de A dans G, et notons A_i la coupe séparant V_i du reste du graphe. Comme $A = A_1 \cup \cdots \cup A_k$ et que chaque arête de A appartient à deux de ces coupes :

$$\sum_{i=1}^{k} w(A_i) = 2w(A).$$

Sans perte de généralité, nous pouvons considérer que A_k est la plus lourde de ces coupes. Il s'agit maintenant de démontrer qu'il existe $k-1$ coupes associées à des arêtes de T dont le poids est inférieur aux coupes $A_1, A_2, \ldots, A_{k-1}$. Le théorème en découlera puisque l'algorithme choisit les $k-1$ coupes les plus légères.

Les $k-1$ coupes sont définies comme suit. Soit B l'ensemble des arêtes de T dont les extrémités appartiennent à deux des $V_1, V_2, \ldots, V_k$ distincts. Étudions le graphe ayant pour sommets V et pour arêtes B. Réduisons chaque ensemble $V_1, V_2, \ldots, V_k$ à un sommet unique. Le graphe réduit est connexe, par connexité de T. Éliminons-en des arêtes jusqu'à obtenir un arbre, et notons $B' \subseteq B$ les arêtes restantes : $|B'| = k-1$. Les arêtes de B' définissent les $k-1$ coupes recherchées.

Maintenant, enracinons cet arbre en V_k et orientons les arêtes vers la racine (rappelons que A_k est la coupe la plus lourde parmi les coupes A_i). Cela va faciliter la description de la correspondance entre les arêtes de B' et les ensembles $V_1, V_2, \ldots, V_{k-1}$: chaque arc est mis en correspondance avec l'ensemble dont il est issu.



Supposons que l'arc $(u, v) \in B'$ soit ainsi mis en correspondance avec l'ensemble V_i. Le poids minimum d'une coupe entre u et v dans G est $w'(u, v)$. Puisque A_i est une coupe entre u et v dans G,

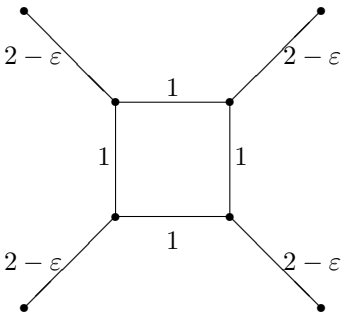
$$w(A_i) \geqslant w'(u, v).$$

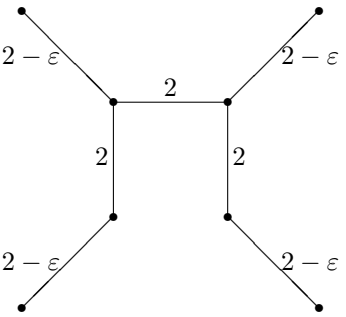
Ainsi chacune des coupes $A_1, A_2, \ldots, A_{k-1}$ est au moins aussi lourde qu'une coupe de G définie par la correspondance des arêtes de B'. En combinant ceci avec le fait que C est l'union des $k-1$ coupes les plus légères définies par T, nous obtenons :

$$w(C) \leqslant \sum_{e \in B'} w'(e) \leqslant \sum_{i=1}^{k-1} w(A_i) \leqslant \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(A_i) = 2\left(1 - \frac{1}{k}\right) w(A).$$

□

Exemple 4.9 L'instance critique pour la coupe multiséparatrice (le graphe à $2k$ sommets) est également une instance critique pour cet algorithme de coupe en k morceaux (bien sûr, il n'est pas nécessaire ici de préciser l'ensemble des sommets terminaux). Les $k-1$ coupes les plus légères dans l'arbre de Gomory-Hu ont chacune pour poids $2 - \varepsilon$, et correspondent aux arêtes de poids $2 - \varepsilon$ dans G. Ainsi, la coupe renvoyée par l'algorithme a pour poids $(k-1)(2-\varepsilon)$. Or, la coupe optimale en k morceaux est l'ensemble des arêtes de poids 1, et son poids est k. Voici le graphe pour $k = 4$ et son arbre de Gomory-Hu :





□

4.3 Exercices

4.1 Montrez comment utiliser l'algorithme 4.3 en tant que sous-programme pour trouver une coupe en k morceaux de poids à un facteur $2 - 2/k$ du minimum. Combien d'appels à ce sous-programme sont nécessaires ?

4.2 Voici un algorithme glouton naturel pour trouver une coupe multiséparatrice. Commençant avec G, calculer des coupes minimales séparant toutes les paires s_i et s_j qui sont encore connectés, puis sélectionner et ôter la plus légère de ces coupes ; itérer jusqu'à ce que toutes les paires s_i et s_j soient déconnectées. Démontrez que cet algorithme est une $(2-2/k)$-approximation.

Les quatre exercices suivants présentent les bases puis un algorithme pour le calcul d'arbre de Gomory-Hu.

4.3 Soit $G = (V, E)$ un graphe non orienté muni d'une fonction de poids positive sur les arêtes $w : E \to \mathbf{R}^+$. Pour $u, v \in V$, notons $f(u, v)$ le poids minimum d'une coupe entre u et v dans G.

1. Soient $u, v, w \in V$ tels que $f(u, v) \leqslant f(u, w) \leqslant f(v, w)$. Démontrez que $f(u, v) = f(u, w)$, c'est-à-dire que les deux plus petites valeurs sont égales.
2. Démontrez que l'ensemble des valeurs de $f(u, v)$ définies par les paires $u, v \in V$, ne contient au plus que $n - 1$ valeurs distinctes.
3. Démontrez que pour tout $u, v, w \in V$,

$$f(u, v) \geqslant \min\{f(u, w), f(w, v)\}.$$

4. Démontrez que pour tout $u, v, w_1, \ldots, w_r \in V$,

$$f(u, v) \geqslant \min\{f(u, w_1), f(w_1, w_2), \ldots, f(w_r, v)\}. \tag{4.1}$$

4.4 Soit T un arbre sur les sommets V, muni d'une fonction de poids sur les arêtes. Nous dirons que T est un *arbre de flot*[7] s'il vérifie la première des conditions de Gomory-Hu, c'est-à-dire si pour toute paire de sommets $u, v \in V$, le poids minimum d'une coupe entre u et v dans G est le même que dans T. Notons K le graphe complet sur V muni de la fonction de poids $f(u, v)$ sur chaque arête uv. Démontrez que tout arbre couvrant de poids maximum de K est un arbre de flot pour G.
Indication : Pour tout $u, v \in V$, considérez l'unique chemin $u, w_1, \ldots, w_r, v$ de u à v dans T. Utilisez (4.1) et le fait que $f(u, v) \leqslant \min\{f(u, w_1), \ldots, f(w_r, v)\}$ lorsque T est un arbre couvrant de poids maximum.

4.5 Soit $(A, \bar{A})$ une coupe de poids minimum entre deux sommets s et t. Considérons le graphe G' obtenu en fusionnant tous les sommets de $\bar{A}$ en un unique sommet $v_{\bar{A}}$. Attribuons aux arêtes $av_{\bar{A}}$ un poids égal à la somme des poids des arêtes ab avec $b \in \bar{A}$. Clairement, toute coupe de G' définit une coupe dans G. Démontrez que pour tous les sommets $x, y \in A$, une coupe de poids minimum entre x et y dans G' définit une coupe de poids minimum entre x et y dans G.

4.6 Nous sommes maintenant prêts à décrire l'algorithme de Gomory et Hu. L'algorithme maintient une partition $\{S_1, S_2, \ldots S_t\}$ de V ainsi qu'un arbre couvrant T ayant pour sommets $\{S_1, \ldots, S_t\}$. T est muni d'une fonction de poids sur les arêtes w', vérifiant l'invariant suivant.
Invariant : Pour toute arête S_iS_j de T, il existe des sommets a et b dans S_i et S_j respectivement, tels que $w'(S_i, S_j) = f(a, b)$; et la coupe associée à l'arête S_iS_j dans T définit une coupe de poids minimum entre a et b dans G.

L'algorithme commence avec la partition triviale $\{V\}$, et exécute $n - 1$ itérations. À chaque itération, l'algorithme sélectionne un ensemble S_i de la partition tel que $|S_i| \geqslant 2$, raffine la partition en divisant S_i, et trouve un arbre vérifiant l'invariant sur cette nouvelle partition. Voici comment il procède.

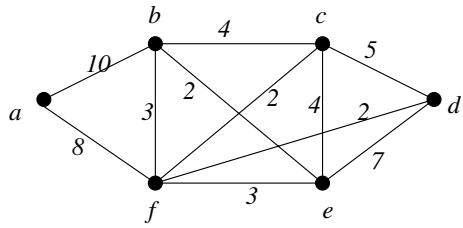
Soient x et y deux sommets distincts de S_i. Enracinons l'arbre courant T en S_i et considérons les sous-arbres enracinés sur les fils de S_i. Réduisons chacun de ces sous-arbres en un unique sommet. Notons G' le graphe réduit (outre les sommets réduits, G' contient tous les sommets de S_i). Calculons une coupe (A, B) de poids minimum entre x et y dans G' (avec $x \in A$ et $y \in B$). Notons w_{xy} le poids de cette coupe. Puis, construisons la nouvelle partition en remplaçant S_i par $S_i^x = S_i \cap A$ et $S_i^y = S_i \cap B$.

L'algorithme met à jour l'arbre de la façon suivante. Le nouvel arbre a une nouvelle arête (S_i^x, S_i^y) de poids w_{xy} ; pour chaque sous-arbre T' qui était incident à T, la racine de T' est connectée à S_i^x ou S_i^y selon que le nœud correspondant à T' était dans A ou dans B, respectivement. Le poids de cette arête est identique à celle qui connectait la racine de T' à S_i avant. Toutes les arêtes internes de T' sont inchangées.

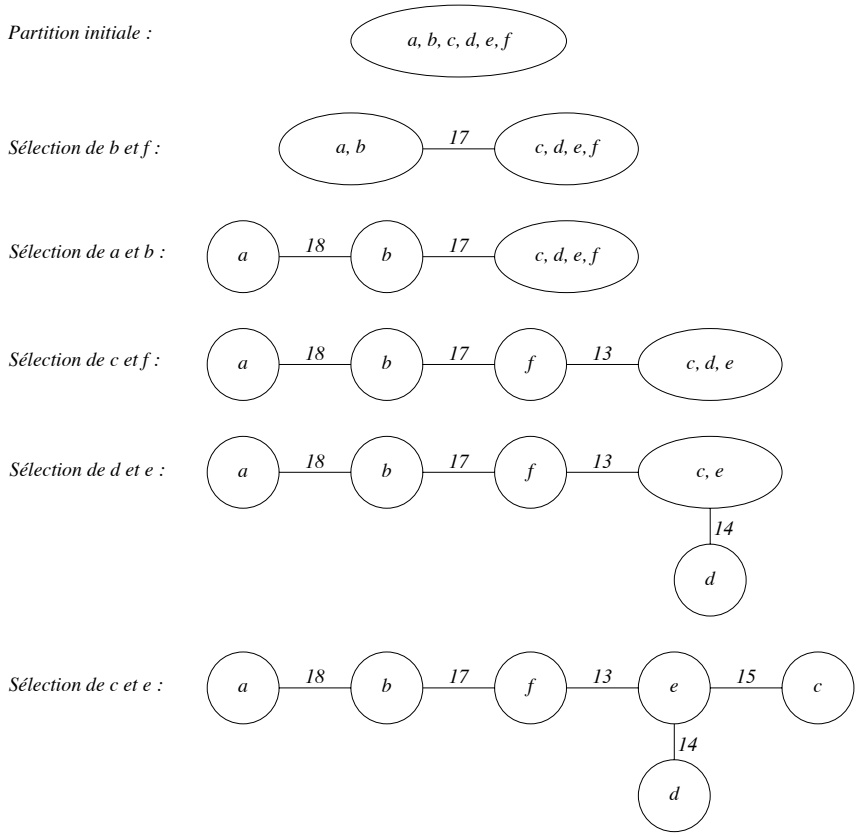
[7] *Flow equivalent tree*, en anglais.

Démontrez que le nouvel arbre vérifie bien l'invariant. En conclure que l'algorithme termine avec un arbre de Gomory-Hu de G (lorsque la partition en contient plus que des singletons).

Exécutons l'algorithme sur le graphe suivant :



L'exécution de l'algorithme de Gomory-Hu donne :



4.7 Démontrer que si l'arbre de Gomory-Hu d'un graphe non orienté G contient $n-1$ poids distincts, alors G admet une unique coupe de poids minimum.

4.4 Notes

L'algorithme 4.3 a été conçu par Dahlhaus, Johnson, Seymour, Papadimitriou et Yannakakis [61]. L'algorithme 4.7 est du à Saran et Vazirani [242] ; la preuve qui en est donnée ici a été proposée par R. Ravi. L'article [117] est l'article original sur les arbres de Gomory-Hu.

5 *k*-Centre

Étudions le problème suivant : étant donné un ensemble de villes dont les distances sont spécifiées, choisir k villes pour y placer des entrepôts de façon à minimiser la distance maximale d'une ville à l'entrepôt le plus proche. Un tel ensemble de k villes est appelé *k-centre*. Nous étudierons en fait la version du problème où les poids sur les arêtes reliant les villes vérifient l'inégalité triangulaire. Sans cette restriction, le problème du k-centre ne peut être approché à un facteur $\alpha(n)$ pour aucune fonction $\alpha(n)$ calculable en temps polynomial, à moins que $\mathbf{P} = \mathbf{NP}$ (voir exercice 5.1).

Pour résoudre ce problème, nous introduirons la méthode de l'*élagage paramétré*.[1] Cette méthode sera utilisée également dans le cadre de la programmation linéaire au chapitre 17.

Problème 5.1 (k-Centre métrique)[2] Soit $G = (V, E)$ un graphe complet non orienté muni d'une fonction de coût métrique sur les arêtes, et k un entier strictement positif. Pour tout sous-ensemble $S \subseteq V$ et tout sommet $v \in V$, notons $\text{connect}(v, S)$ le coût minimal d'une arête reliant v à un sommet de S. Trouver un sous-ensemble $S \subseteq V$ tel que $|S| = k$ et qui minimise : $\max_v\{\text{connect}(v, S)\}$.

5.1 Élagage paramétré appliqué au *k*-centre métrique

Si nous connaissions le coût d'une solution optimale, nous pourrions éliminer une partie des entrées inutile à la résolution du problème et donc simplifier la recherche de la bonne solution. Cependant, comme nous l'avons vu au chapitre 1, calculer le coût d'une solution optimale est justement le cœur d'un problème d'optimisation **NP**-difficile. L'élagage paramétré permet de contourner cette difficulté. L'idée est de fixer un paramètre t, qui est notre « hypothèse » sur la valeur du coût optimal. Pour chaque valeur de t, l'instance I est élaguée en éliminant les parties qui ne sont utilisées par aucune solution de coût $\leqslant t$. Notons $I(t)$ l'instance élaguée. L'algorithme procède en deux étapes. La première étape calcule un minorant t^* de OPT à

[1] *Parametric pruning*, en anglais.
[2] *Metric k-center*, en anglais.

partir de la famille des $I(t)$. La seconde étape recherche une solution d'une instance $I(\alpha\, t^*)$, pour un choix adéquat de α.

L'élagage paramétré s'applique naturellement au problème du k-centre en le reformulant comme suit. Commençons par trier les arêtes de G par coût croissant, c'est-à-dire $\text{coût}(e_1) \leqslant \text{coût}(e_2) \leqslant \ldots \leqslant \text{coût}(e_m)$ et notons $G_i = (V, E_i)$, où $E_i = \{e_1, e_2, \ldots, e_i\}$. Rappelons qu'un *dominant* d'un graphe non orienté $H = (U, F)$ est un sous-ensemble de sommets $S \subseteq U$ tels que tout sommet de $U \setminus S$ est adjacent à un sommet de S. Notons $\text{dom}(H)$ la taille d'un plus petit dominant de H. Le calcul de $\text{dom}(H)$ est **NP**-difficile. Trouver un k-centre est en fait équivalent à trouver le plus petit index i tel que G_i admet un dominant de taille au plus k, c'est-à-dire tel que G_i se décompose en au plus k étoiles couvrant tous ses sommets. Remarquons également que si i^* est ce plus petit indice, $\text{coût}(e_{i^*})$ est le coût d'un k-centre optimal. Nous le noterons OPT. Nous travaillerons désormais sur la famille de graphes $G_1, \ldots, G_m$.

Le *carré d'un graphe* H, noté H^2, est défini comme le graphe qui contient une arête uv pour chaque paire de sommets u et v reliés par un chemin de longueur au plus 2 dans H. Le résultat structurel suivant permet de minorer OPT.

Lemme 5.2 *Étant donné un graphe H et I un stable*[3] *de H^2, alors :* $|I| \leqslant \text{dom}(H)$.

Preuve : Soit D un dominant minimum de H. H se décompose en $|D|$ étoiles couvrant tous les sommets de H. Puisqu'une étoile de H est une clique de H^2, H^2 se décompose en au plus $|D|$ cliques couvrant tous ses sommets. Par conséquent, tout stable I de H^2 ne pourra choisir qu'un unique sommet dans chacune de ces cliques, CQFD. □

Voici l'algorithme du k-centre :

Algorithme 5.3 (k-Centre métrique)

1. Construire $G_1^2, G_2^2, \ldots, G_m^2$.
2. Calculer un stable maximal, M_i, pour chaque G_i^2.
3. Calculer le plus petit indice i tel que $|M_i| \leqslant k$. Notons-le j.
4. Renvoyer M_j.

L'algorithme est construit sur le minorant suivant :

Lemme 5.4 *Soit j l'indice calculé par l'algorithme, alors :* $\text{coût}(e_j) \leqslant \text{OPT}$.

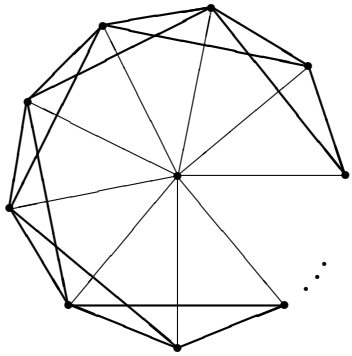
[3] Un *stable* (*Independent set*, en anglais) d'un graphe $G = (V, E)$ est un sous-ensemble de sommets $I \subseteq V$ totalement déconnecté, c'est-à-dire tel que pour tout $u, v \in I$, $u \neq v \Rightarrow uv \notin E$.

Preuve : Pour tout $i < j$, $|M_i| > k$. Or, le lemme 5.2 donne que $\mathrm{dom}(G_i) > k$, et donc que $i^* > i$. Par conséquent, $j \leqslant i^*$ et $\mathrm{coût}(e_j) \leq \mathrm{coût}(e_{i^*})$. □

Théorème 5.5 *L'algorithme 5.3 est une 2-approximation pour le problème du k-centre métrique.*

Preuve : L'observation clé est qu'un stable maximal I d'un graphe en est également un dominant. En effet, si un sommet v n'était pas dominé par I, $I \cup \{v\}$ serait un stable, contredisant ainsi la maximalité de I. Il existe donc dans G_j^2 des étoiles centrées sur les sommets de M_j qui couvrent tous les sommets. Or, l'inégalité triangulaire donne que toutes les arêtes de G_j utilisées par ces étoiles ont un coût $\leqslant 2 \cdot \mathrm{coût}(e_j)$. Appliquer le lemme 5.4 termine alors la preuve du théorème. □

Exemple 5.6 Une instance critique pour l'algorithme 5.3 est donnée par le graphe suivant en forme de roue à $n + 1$ sommets. Toutes les arêtes incidentes sur le sommet central coûtent 1, et les autres coûtent 2 :



Le graphe représenté ici est G_{3n} ; pour obtenir le graphe complet donné en entrée, il suffit d'ajouter toutes les arêtes manquantes et de leur donner un coût 2 ; les arêtes fines coûtent 1 ; les arêtes en gras coûtent 2 — les arêtes de coût 2 ne sont pas toutes représentées.

Pour $k = 1$, la solution optimale est le centre de la roue et $\mathrm{OPT} = 1$. Cependant, l'algorithme aboutit à $j = n$. Or G_n^2 est une clique, et si un nœud périphérique est choisi pour construire le stable maximal, alors le coût de la solution sera de 2. □

Nous allons démontrer maintenant que 2 est le meilleur facteur d'approximation qu'on puisse espérer pour le problème du k-centre métrique.

Théorème 5.7 *Si* $\mathbf{P} \neq \mathbf{NP}$*, quel que soit $\varepsilon > 0$, il n'existe pas de $(2 - \varepsilon)$-approximation pour le problème du k-centre métrique.*

Preuve : Nous allons démontrer qu'avec un tel algorithme, nous pouvons résoudre le problème du dominant minimum en temps polynomial. L'idée de la preuve est similaire à celle du théorème 3.6. Il s'agit de réduire le problème du dominant minimum à celui de l'approximation du k-centre métrique. Considérons $G = (V, E)$ et k, une instance du problème du dominant minimum. Nous construisons le graphe complet $G' = (V, E')$ muni de la fonction de coût sur les arêtes suivantes :

$$\text{coût}(u, v) = \begin{cases} 1, \text{ si } (u, v) \in E, \\ 2, \text{ si } (u, v) \notin E. \end{cases}$$

La fonction de coût sur G' vérifie trivialement l'inégalité triangulaire (car les coûts des arêtes valent 1 ou 2). Cette réduction vérifie les propriétés suivantes :

- si $\text{dom}(G) \leqslant k$, alors G' admet un k-centre de coût 1 ; et
- si $\text{dom}(G) > k$, alors le coût optimal d'un k-centre de G' vaut 2.

Dans le premier cas, une $(2 - \varepsilon)$-approximation pour le k-centre donne nécessairement une solution de coût 1, puisqu'elle ne peut utiliser d'arête de coût 2. Par conséquent, utiliser un tel algorithme permettrait de distinguer entre ces deux possibilités, et de résoudre le problème du dominant minimum. □

5.2 k-Centre métrique pondéré

Nous allons appliquer la technique de l'élagage paramétré et obtenir une 3-approximation pour la généralisation suivante du problème du k-centre métrique.

Problème 5.8 (k-centre métrique pondéré)[4] L'entrée consiste en une fonction de poids $w : V \to R^+$ et une borne $W \in R^+$ en sus de la fonction de coût sur les arêtes. Le problème est de trouver un sous-ensemble $S \subseteq V$ de poids total $\leqslant W$ minimisant la même fonction objectif que précédemment, c'est-à-dire :

$$\max_{v \in V} \{\min_{u \in S} \{\text{coût}(u, v)\}\}.$$

Notons $\text{poidsdom}(G)$ le poids minimum d'un dominant de G. En définissant la famille de graphes (G_i) comme précédemment, nous sommes ramenés à trouver le plus petit indice i tel que $\text{poidsdom}(G_i) \leq W$. Soit i^* cet indice. Le coût d'une solution optimale vaut $\text{OPT} = \text{coût}(e_{i^*})$.

Soit H, un graphe pondéré. Considérons un stable I de H^2. Pour chaque $u \in I$, notons $s(u)$ le sommet le plus léger parmi u et ses voisins dans H

[4] *Metric weighted k-center*, en anglais.

(et non dans H^2). Posons $S = \{s(u)|\ u \in I\}$. Le fait suivant, analogue du lemme 5.2, nous permettra de minorer OPT :

Lemme 5.9 $w(S) \leq \text{poidsdom}(H)$.

Preuve : Soit D un dominant de poids minimum dans H. Il existe, par définition de D, un ensemble d'étoiles disjointes de H, centrées sur les sommets de D et couvrant la totalité des sommets de H. Puisque chacune de ces étoiles est une clique dans H^2, le stable I de H^2 contient au plus un sommet de chaque étoile. Or, chaque sommet de I est soit dans D, soit adjacent à un sommet de D dans H, nous en concluons que : $w(S) \leqslant w(D)$. □

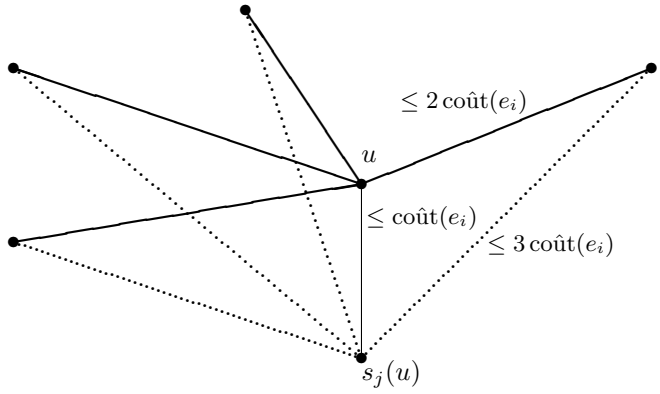
Voici l'algorithme. Notons $s_i(u)$ le sommet le plus léger parmi u et ses voisins dans G_i.

Algorithme 5.10 (k-Centre métrique pondéré)

1. Construire $G_1^2, G_2^2, \ldots, G_m^2$.
2. Calculer un stable maximal, M_i, pour chaque G_i^2.
3. Calculer $S_i = \{s_i(u)|\ u \in M_i\}$.
4. Trouver l'index minimal i tel que $w(S_i) \leqslant W$. Notons-le j.
5. Renvoyer S_j.

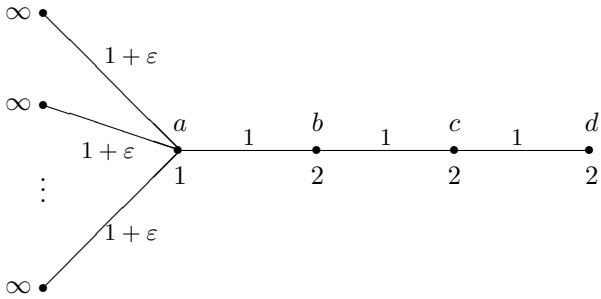
Théorème 5.11 *L'algorithme 5.10 est une 3-approximation pour le problème du k-centre métrique pondéré.*

Preuve : D'après le lemme 5.9 et par un argument similaire à celui du lemme 5.4, OPT est minoré par coût(e_j). Puisque M_j est aussi un dominant de G_j^2, nous pouvons couvrir V avec des étoiles de G_j^2 centrées sur les sommets de M_j. L'inégalité triangulaire donne que chaque arête de ces étoiles coûte au plus $2 \cdot$ coût(e_j).



Tous les centres des étoiles sont reliés à un sommet de S_j par une arête de coût $\leq$ coût(e_j). Nous recentrons chaque étoile sur le sommet de S_j auquel son centre était adjacent. L'inégalité triangulaire donne que le coût de l'arête la plus coûteuse utilisée par cette construction est majoré par $3 \cdot$ coût(e_j). □

Exemple 5.12 Le graphe suivant à $n+4$ sommets est une instance critique pour cet algorithme. Les poids des sommets et des arêtes sont donnés sur la figure ; les coûts des arêtes manquantes sont donnés par ceux des plus courts chemins.



Il est facile de voir que pour $W = 3$, le coût optimal d'un k-centre est $1+\varepsilon$: $\{a, c\}$ est un k-centre optimal. Or, pour tout $i < n+3$, l'ensemble S_i calculé par l'algorithme contient un sommet de poids infini. Imaginons que, pour $i = n+3$, l'algorithme choisisse $M_{n+3} = \{b\}$ comme stable maximal de G^2_{n+3}. Alors, $S_{n+3} = \{a\}$ est la réponse de l'algorithme et coûte 3. □

5.3 Exercices

5.1 Démontrez que si les coûts sur des arêtes ne vérifient pas l'inégalité triangulaire, alors le problème du k-centre ne peut être approché à un facteur $\alpha(n)$ pour aucune fonction $\alpha(n)$ calculable en temps polynomial.
Indication : Réunissez les idées des théorèmes 3.6 et 5.7.

5.2 Étudions l'étape 2 de l'algorithme 5.3, qui calcule un stable maximal de G^2_i. Un choix sans doute plus naturel aurait été de calculer un dominant minimal. Modifiez l'algorithme 5.3 pour qu'il choisisse un dominant minimal de G^2_i. Démontrez que l'algorithme obtenu n'est pas une 2-approximation. Quel facteur d'approximation pouvez-vous garantir pour cet algorithme ?
Indication : Suite à cette modification, le minorant n'est plus utilisable, car les conditions du lemme 5.2 ne sont plus vérifiées si I est un dominant minimal (seulement) de H^2.

5.3 (Gonzalez [118]) Étudions le problème suivant.

Problème 5.13 (k-classification métrique)[5] Soit $G = (V, E)$ un graphe complet non orienté muni d'une fonction de coût métrique sur les arêtes, et k un entier strictement positif. Trouver une partition de V en $V_1, \ldots, V_k$ de façon à minimiser le coût maximal d'une arête entre deux sommets d'une même composante, c'est-à-dire de façon à minimiser :

$$\max_{1 \leqslant i \leqslant k,\ u,v \in V_i} \text{coût}(u, v).$$

1. Proposez une 2-approximation pour ce problème, ainsi qu'une instance critique.
2. Démontrez que ce problème n'admet de $(2-\varepsilon)$-approximation pour aucun $\varepsilon > 0$, à moins que $\mathbf{P} = \mathbf{NP}$.

5.4 (Khuller, Pless, et Sussmann [177]) Il existe une variante robuste du problème du k-centre incluant une tolérance aux pannes. Cette variante admet une entrée supplémentaire $\alpha \leqslant k$, indiquant le nombre de centres auxquels chaque ville doit être connectée chaque ville. Il s'agit toujours de sélectionner k centres qui minimisent la distance de l'arête la plus longue d'une ville aux α centres les plus proches.

On dira qu'un sous-ensemble $S \subseteq V$ d'un graphe non orienté $H = (V, E)$ est α-*dominant*[6] si tout sommet $v \in V$ est adjacent à au moins α sommets de S (en considérant qu'un sommet est toujours adjacent à lui-même). Notons $\text{dom}_\alpha(H)$ la taille minimale d'un α-dominant de H.

1. Soit I un stable de H^2. Démontrez que $\alpha|I| \leqslant \text{dom}_\alpha(H)$.
2. Proposez une 3-approximation pour le problème du k-centre robuste.
   Indication : Calculez un stable maximal M_i de G_i^2, pour $1 \leqslant i \leqslant m$. Recherchez le plus petit indice i tel que $|M_i| \leqslant \lfloor \frac{k}{\alpha} \rfloor$, et tel que le degré de chaque sommet de M_i dans G_i soit $\geqslant \alpha - 1$.

5.5 (Khuller, Pless, et Sussmann [177]) Considérons la variante suivante du problème de l'exercice 5.4, dans laquelle les sommets de S n'ont pas de contrainte de connectivité et où seuls ceux de $V - S$ doivent satisfaire ces contraintes. Tout sommet de $V - S$ doit être connecté à α sommets de S. Le but est de nouveau de trouver S tel que $|S| = k$ et qui minimise la longueur de l'arête la plus longue utilisée.

L'algorithme développé pour ce problème considère chaque graphe G_i. Il commence avec $S_i = \varnothing$. Nous dirons qu'un sommet $v \in V \setminus S_i$ est *connecté* j fois s'il est adjacent à j sommets de S_i par des arêtes de G_i^2. Tant qu'il existe un sommet $v \in V \setminus S_i$ qui n'est pas connecté α fois, l'algorithme sélectionne le sommet le moins connecté et l'ajoute à S_i. Enfin, il trouve l'index minimal tel que $|S_i| \leqslant k$, notons-le j, et renvoie S_j.

Démontrez que cet algorithme est une 2-approximation.

[5] *Metric k-cluster*, en anglais.
[6] α-*Dominating set*, en anglais.

5.4 Notes

Les deux algorithmes présentés dans ce chapitre pour le k-centre sont dus à Hochbaum et Shmoys [135]. Le théorème 5.7 est quant à lui dû à Hsu et Nemhauser [140].

6 Coupe-cycles de sommets

Nous allons utiliser dans ce chapitre la technique du *mille-feuille*, introduite au chapitre 2, afin d'obtenir une 2-approximation pour le problème suivant. Rappelons que le principe du mille-feuille est de décomposer la fonction de poids en une somme de fonctions de poids plus pratiques définies sur une suite de sous-graphes imbriqués de G.

Problème 6.1 (Coupe-cycles de sommets minimum)[1] Étant donné un graphe non orienté $G = (V, E)$ muni d'une fonction de poids w positive sur les sommets, trouver un sous-ensemble de V de poids minimum dont le retrait de G engendre un graphe acyclique.

6.1 Graphes pondérés cyclomatiques

Donnons-nous un ordre arbitraire sur les arêtes de G. Associons à chaque cycle simple C de G, son *vecteur caractéristique* : il s'agit du vecteur de $\mathrm{GF}[2]^m$ (où $m = |E|$) dont la i-ième coordonnée vaut 1 ou 0 suivant que la i-ième arête de E appartient à C ou non, respectivement. L'*espace des cycles de* G est le sous-espace de $\mathrm{GF}[2]^m$ engendré par les vecteurs caractéristiques des cycles simples de G. La dimension de cet espace est appelé le *nombre cyclomatique de* G, noté $\mathrm{cyc}(G)$. Nous noterons $\mathrm{comps}(G)$ le nombre de composantes connexes de G.

Théorème 6.2 $\mathrm{cyc}(G) = |E| - |V| + \mathrm{comps}(G)$.

Preuve : L'espace des cycles d'un graphe est la somme directe des espaces des cycles de ces composantes connexes. Ainsi le nombre cyclomatique de G est la somme des nombres cyclomatiques des composantes connexes. Il suffit donc de prouver le théorème pour un graphe connexe G.

Soit T un arbre couvrant de G. Nous associons à chaque arête e qui n'est pas dans T son *cycle fondamental* : l'unique cycle présent dans $T \cup \{e\}$. L'ensemble des vecteurs caractéristiques de ces cycles forme une famille libre (chaque cycle possède une arête qui n'appartient à aucun autre). Par conséquent, $\mathrm{cyc}(G) \geq |E| - |T| = |E| - |V| + 1$.

[1] *Feedback vertex set problem*, en anglais.

Chaque arête e de T définit une *coupe fondamentale*, $(S, \overline{S})$ de G, $S \subset V$ (S et $\overline{S}$ sont les deux composantes connexes engendrées par le retrait de e dans T). Nous associons à chaque coupe son *vecteur caractéristique* dans $\mathrm{GF}[2]^m$, dont la i-ième coordonnée vaut 1 ou 0 suivant que la i-ième arête appartient à la coupe ou non, respectivement. Considérons les $|V|-1$ vecteurs définis par les arêtes de T. Puisque tout cycle traverse un nombre pair de fois chaque coupe, ces vecteurs sont orthogonaux à l'espace des cycles de G. Ces $|V|-1$ vecteurs forment une famille libre, car chaque coupe possède une arête qui n'appartient à aucune des $|V|-2$ autres coupes (l'arête de l'arbre définissant cette coupe). Par conséquent, la dimension de l'orthogonal de l'espace des cycles est minorée par $|V|-1$. Il en résulte que $\mathrm{cyc}(G) \leq |E|-|V|+1$. Ainsi par l'inégalité précédente : $\mathrm{cyc}(G) = |E|-|V|+1$. □

Notons $\delta_G(v)$ la variation du nombre cyclomatique du graphe lors du retrait d'un sommet v. Le retrait d'un coupe-cycles de sommets $F = \{v_1, \ldots, v_f\}$ annule le nombre cyclomatique de G ; ainsi :

$$\mathrm{cyc}(G) = \sum_{i=1}^{f} \delta_{G_{i-1}}(v_i),$$

avec $G_0 = G$ et, pour $i > 0$, $G_i = G \setminus \{v_1, \ldots, v_i\}$. Le lemme 6.4 ci-après nous donnera :

$$\mathrm{cyc}(G) \leq \sum_{v \in F} \delta_G(v). \tag{6.1}$$

Une fonction de poids est *cyclomatique* s'il existe une constante $c > 0$ telle que le poids de chaque sommet v vaille $c \cdot \delta_G(v)$. Soient w une fonction de poids cyclomatique et F un coupe-cycles optimal. L'inégalité (6.1) donne

$$c \cdot \mathrm{cyc}(G) \leqslant c \sum_{v \in F} \delta_G(v) = w(F) = \mathrm{OPT},$$

et donc $c \cdot \mathrm{cyc}(G)$ minore OPT. L'intérêt des fonctions de poids cyclomatiques apparaît au lemme 6.5 plus bas : pour toute fonction de poids cyclomatique, tout coupe-cycles minimal a un poids inférieur au double du poids optimal.

Notons $\deg_G(v)$, le degré de v dans G, et $\mathrm{comps}(G \setminus v)$ le nombre de composantes connexes engendrées par le retrait de v dans G. Le fait suivant est une conséquence immédiate du théorème 6.2 appliqué à G et $G \setminus v$.

Fait 6.3 *Pour tout graphe connexe G, $\delta_G(v) = \deg_G(v) - \mathrm{comps}(G \setminus v)$.*

Lemme 6.4 *Soit H un sous-graphe de G (pas nécessairement induit par des sommets). Alors, $\delta_H(v) \leq \delta_G(v)$.*

Preuve : Il suffit de démontrer le lemme pour chaque composante connexe de G et uniquement lorsque v est un sommet de H. Nous supposerons donc que

G et H sont connexes (H ne contenant pas nécessairement tous les sommets de G). D'après le fait 6.3, il s'agit de démontrer l'inégalité :

$$\deg_H(v) - \mathrm{comps}(H \setminus v) \leqslant \deg_G(v) - \mathrm{comps}(G \setminus v).$$

Démontrons que l'ajout d'arêtes de $G \setminus H$ ne peut qu'augmenter la quantité $\deg_H(v) - \mathrm{comps}(H \setminus v)$; une récurrence immédiate terminera la preuve. Considérons les composantes connexes engendrées par le retrait de v dans H. Les arêtes de $G \setminus H$ qui ne sont pas incidentes à v, ne peuvent que réduire le nombre de ces composantes (et ne changent pas le degré de v). L'ajout d'une arête de $G \setminus H$ incidente à v, peut engendrer une nouvelle composante, mais cet ajout est compensé par la contribution de cette arête au degré de v. □

Lemme 6.5 *Si F est un coupe-cycles minimal de sommets de G, alors*

$$\sum_{v \in F} \delta_G(v) \leq 2 \cdot \mathrm{cyc}(G).$$

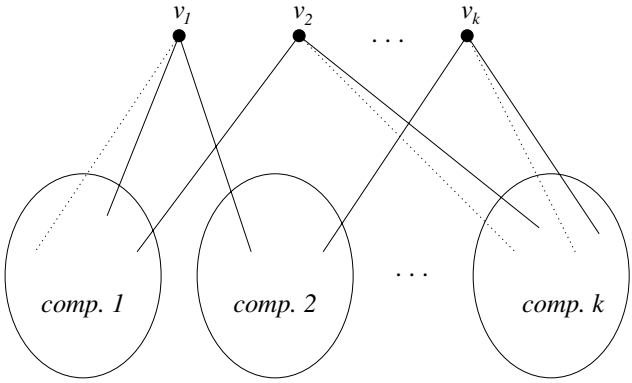
Preuve : Comme l'espace des cycles de G est la somme directe des espaces des cycles de ses composantes connexes, il suffit de prouver ce lemme pour tout graphe G connexe.

Posons $F = \{v_1, \ldots, v_f\}$, et notons k le nombre de composantes connexes engendrées en ôtant F de G. Classons ces composantes en deux catégories : celles dont les arêtes sont incidentes à un unique sommet de F, et celles dont les arêtes sont incidentes à au moins deux sommets de F. Soient t et $k-t$ les nombres de composantes dans chaque catégorie respectivement. Nous allons démontrer que

$$\sum_{i=1}^{f} \delta_G(v_i) = \sum_{i=1}^{f} (\deg_G(v_i) - \mathrm{comps}(G \setminus v_i)) \leqslant 2(|E| - |V|),$$

terminant ainsi la preuve du lemme. Clairement, $\sum_{i=1}^{f} \mathrm{comps}(G \setminus v_i) \geqslant f + t$. Par conséquent, il nous reste à montrer que

$$\sum_{i=1}^{f} \deg_G(v_i) \leqslant 2(|E| - |V|) + f + t.$$



Comme F est un coupe-cycles, chacune des k composantes est un arbre. Le nombre total d'arêtes dans ces composantes est donc $|V|-f-k$. Minorons à present le nombre d'arêtes de la coupe $(F, V-F)$. Puisque F est minimal, chaque $v_i \in F$ appartient à un cycle qui ne contient aucun autre sommet de F. Par conséquent, chaque v_i appartient à au moins deux arêtes incidentes à l'une des composantes. Pour chaque v_i, ôtons arbitrairement l'une de ces arêtes dans G ; soit au total f arêtes. Chacune des t composantes compte maintenant au moins une arête incidente à F, et au moins deux pour chacune des $k-t$ autres. Le nombre d'arêtes dans la coupe $(F, V \setminus F)$ est donc minoré par $f+t+2(k-t) = f+2k-t$.

Nous tirons de ces deux faits que :

$$\sum_{i=1}^{f} \deg_G(v_i) \leqslant 2|E| - 2(|V| - f - k) - (f + 2k - t),$$

CQFD. □

Corollaire 6.6 *Étant donné une fonction de poids cyclomatique w sur les sommets de G, et F un coupe-cycles minimal de sommets de G, alors $w(F) \leq 2 \cdot \mathrm{OPT}$.*

6.2 Coupe-cycles par la technique du mille-feuille

Étudions à présent les graphes pondérés arbitraires. Considérons l'opération élémentaire suivante. Étant donné un graphe $G = (V, E)$ muni d'une fonction de poids w sur les sommets, posons

$$c = \min_{v \in V} \left\{ \frac{w(v)}{\delta_G(v)} \right\}.$$

La fonction de poids $t(v) = c \cdot \delta_G(v)$ est la *plus grande fonction de poids cyclomatique sous w*. Notons $w'(v) = w(v) - t(v)$ la *fonction de poids résiduelle*,

puis V' l'ensemble des sommets de poids résiduels non nuls (clairement $V' \subset V$), et G' le sous-graphe de G induit par V'.

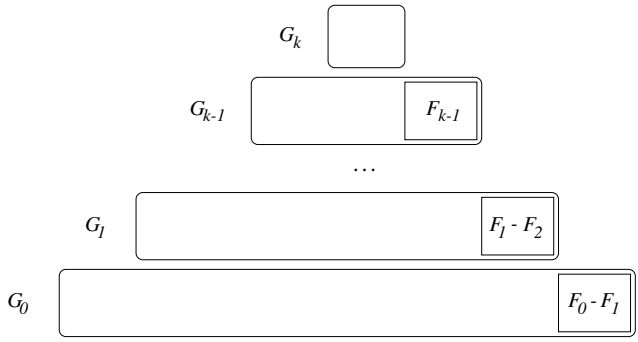
Cette opération élémentaire permet de décomposer G en une suite décroissante de sous-graphes induits, qui se termine avec un graphe acyclique (chaque opération soustrait la plus grande fonction de poids cyclomatique sous la fonction de poids résiduelle courante). Notons $G = G_0 \supset G_1 \supset \cdots \supset G_k$ cette suite de graphes, où G_k est acyclique ; G_i est le sous-graphe de G induit par un ensemble de sommets V_i, avec $V = V_0 \supset V_1 \supset \cdots \supset V_k$. Notons t_i, $i = 0, \ldots, k-1$, la fonction de poids cyclomatique définie sur G_i. La fonction de poids résiduelle sur G_0 est $w_0 = w$, t_0 est la plus grande fonction de poids cyclomatique sous w_0, $w_1 = w_0 - t_0$ est la fonction de poids résiduelle sur G_1, ainsi de suite. Enfin, w_k est la fonction de poids résiduelle sur G_k. Posons $t_k = w_k$ pour simplifier les notations. Le poids d'un sommet v se décompose sur $t_0, t_1, \ldots, t_k$, donc

$$\sum_{i:\, v \in V_i} t_i(v) = w(v).$$

Le lemme suivant va nous donner un algorithme pour construire un coupe-cycles de sommets auquel nous pourrons appliquer le lemme 6.5.

Lemme 6.7 *Soit H un sous-graphe de $G = (V, E)$ induit par $V' \subset V$. Soient F un coupe-cycles minimal de sommets pour H et $F' \subseteq V \setminus V'$ un ensemble minimal tel que $F \cup F'$ soit un coupe-cycles pour G. Alors $F \cup F'$ est un coupe-cycles minimal de G.*

Preuve : Comme F est minimal pour H, nous pouvons associer à tout $v \in F$, un cycle C de H qui ne passe par aucun autre sommet de F. Puisque $F' \cap V' = \varnothing$, C passe par un unique sommet de $F \cup F'$, v, et donc v n'est pas redondant. □



À la fin de la décomposition, $F_k = \varnothing$ est un coupe-cycles minimal de G_k. Pour $i = k, k-1, \ldots, 1$, le coupe-cycles minimal F_i trouvé dans G_i est augmenté de façon minimale par des sommets de $V_{i-1} \setminus V_i$ pour donner un coupe-cycles minimal F_{i-1} de G_{i-1}. Le dernier ensemble, F_0, est un

coupe-cycles minimal de G.

Théorème 6.8 *L'algorithme 6.9 est une* 2*-approximation pour le problème du coupe-cycles minimum de sommets.*

Algorithme 6.9 (coupe-cycles de sommets)

1. Décomposition

   $H \leftarrow G$, $w' \leftarrow w$, $i \leftarrow 0$

   Tant que H n'est pas acyclique,

   $c \leftarrow \min_{u \in H} \left\{ \frac{w'(u)}{\delta_H(u)} \right\}$

   $G_i \leftarrow H$, $t_i \leftarrow c \cdot \delta_{G_i}$, $w' \leftarrow w' - t_i$

   $H \leftarrow$ le sous-graphe de G_i induit par les sommets u tels que $w'(u) > 0$

   $i \leftarrow i + 1$.

   $k \leftarrow i$, $G_k \leftarrow H$

2. Augmentation

   $F_k \leftarrow \varnothing$

   Pour $i = k, \ldots, 1$, calculer un coupe-cycles F_{i-1} de G_{i-1} en augmentant F_i d'un ensemble minimal de sommets choisis dans $V_{i-1} \setminus V_i$.

3. Renvoyer F_0.

Preuve : Soit F^* un coupe-cycles de sommets optimal pour G. Comme G_i est un sous-graphe induit de G, $F^* \cap V_i$ est également un coupe-cycles de G_i (pas nécessairement optimal). Puisque les poids de sommets se réécrivent comme la somme des fonctions t_i, nous avons

$$\mathrm{OPT} = w(F^*) = \sum_{i=0}^{k} t_i(F^* \cap V_i) \geq \sum_{i=0}^{k} \mathrm{OPT}_i,$$

où OPT_i est le poids d'un coupe-cycles de sommets optimal de G_i pour la fonction de poids t_i.

Décomposons le poids de F_0 :

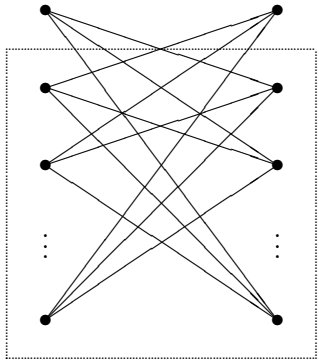
$$w(F_0) = \sum_{i=0}^{k} t_i(F_0 \cap V_i) = \sum_{i=0}^{k} t_i(F_i).$$

F_i est un coupe-cycles minimal de G_i d'après le lemme 6.7. Or, pour $0 \leqslant i \leqslant k-1$, le fonction de poids t_i est cyclomatique. Le lemme 6.5 implique donc que $t_i(F_i) \leqslant 2\,\mathrm{OPT}_i$ (rappelons que $F_k = \varnothing$). Ainsi,

$$w(F_0) \leqslant 2\sum_{i=0}^{k} \mathrm{OPT}_i \leqslant 2 \cdot \mathrm{OPT}\,.$$

□

Exemple 6.10 Considérons le graphe obtenu en ôtant un couplage parfait d'un graphe biparti complet, puis en dédoublant les arêtes restantes. C'est une instance critique pour cet algorithme — remarquez que l'algorithme reste valide en présence d'arêtes multiples, et que pour obtenir une instance critique sans arêtes multiples, il suffit de placer un sommet très lourd sur chaque arête.



Supposons que la fonction de poids du graphe soit cyclomatique, tous les sommets ont le même poids. La phase de décomposition de l'algorithme ne produit donc qu'un unique graphe non trivial, G lui-même, pour lequel l'algorithme calcule un coupe-cycles de sommets minimal. Les sommets dans les pointillés forment donc une sortie possible de l'algorithme. Or, cet ensemble compte $2n-2$ sommets alors qu'un coupe-cycles minimum, un des côtés de la bipartition, n'en compte que n. □

6.3 Exercices

6.1 Un algorithme glouton naturel pour trouver un coupe-cycles de sommets consiste à sélectionner et à ôter itérativement le sommet de meilleur coût efficace, c'est-à-dire le sommet v qui minimise $w(v)/\delta_H(v)$ (où H est le graphe courant), et ce jusqu'à ce que tous les cycles aient disparu. Donnez

des exemples démontrant que cet algorithme n'est pas une approximation à un facteur constant. Quel est le facteur d'approximation garanti par cet algorithme ?

6.2 Exhibez une réduction isofacteur du problème de la couverture par sommets au problème du coupe-cycles de sommets (prouvant ainsi qu'améliorer le facteur d'approximation du second améliorerait celui du premier ; se reporter à la section 30.1).

6.4 Notes

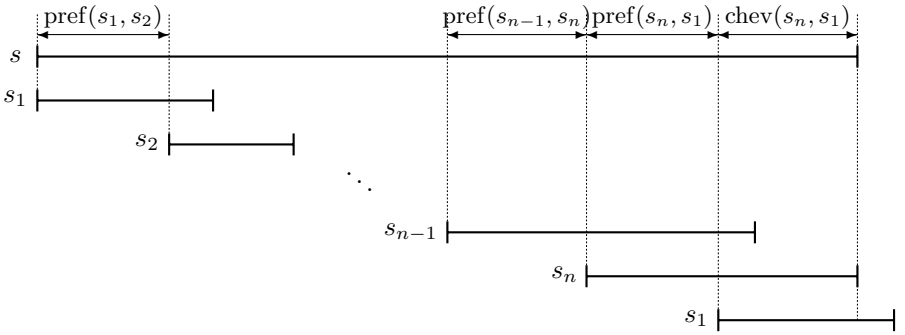
L'algorithme 6.9 est dû à Bafna, Berman, et Fujito [21] (voir également Becker et Geiger [25] et Chudak, Goemans, Hochbaum, et Williamson [49] pour d'autres 2-approximations pour le problème du coupe-cycles de sommets minimum).

7 Surfacteur minimum

Le problème du surfacteur minimum[1] a été défini au chapitre 2 (problème 2.9). Nous y avons proposé une première approximation fondée sur la couverture par ensembles. Dans ce chapitre, nous donnons d'abord une 4-approximation, que nous améliorons ensuite pour obtenir une 3-approximation.

7.1 Une 4-approximation

Commençons par exhiber un bon minorant de OPT. Supposons que les mots $s_1, s_2, \ldots, s_n$ sont numérotés par ordre de première occurrence à gauche dans un surfacteur minimum s.



Notons chevauch(s_i, s_j), le *chevauchement*[2] de s_i sur s_j, c'est-à-dire le plus long suffixe de s_i qui soit un préfixe de s_j, et préfixe(s_i, s_j), le préfixe de s_i obtenu en éliminant son chevauchement avec s_j. Par minimalité du surfacteur s, le recouvrement de deux s_i consécutifs dans s est toujours maximum. Ainsi, en supposant qu'aucun des s_i n'est facteur d'un autre, nous obtenons

[1] *Shortest superstring*, en anglais.
[2] *Overlap*, en anglais.

$$\begin{aligned}\mathrm{OPT} = {} & |\,\text{préfixe}(s_1, s_2)| + |\,\text{préfixe}(s_2, s_3)| + \ldots + |\,\text{préfixe}(s_n, s_1)| \\ & + |\,\text{chevauch}(s_n, s_1)|. \end{aligned} \tag{7.1}$$

Remarquons que l'on a répété s_1 à la fin pour obtenir les deux derniers termes dans (7.1). Cette égalité évoque un lien étroit entre le problème du surfacteur minimum pour S et celui du voyageur de commerce dans le *graphe des préfixes de* S, c'est-à-dire le graphe orienté complet dont les sommets sont $\{1, \ldots, n\}$ et les arcs $i \to j$ ont pour poids $|\,\text{préfixe}(s_i, s_j)|$ pour tout $i \neq j$ — les boucles sont exclues. Dans ce graphe, $|\,\text{préfixe}(s_1, s_2)| + |\,\text{préfixe}(s_2, s_3)| + \ldots + |\,\text{préfixe}(s_n, s_1)|$ est le poids du circuit $1 \to 2 \to \ldots \to n \to 1$. D'après (7.1), le poids optimal d'un circuit du voyageur de commerce dans le graphe des préfixes est donc un minorant de OPT. En tant que tel, ce minorant n'est pas très utile, car nous ne pouvons pas calculer efficacement un circuit optimal pour le voyageur de commerce.

La clé pour minorer OPT est d'utiliser une *couverture par circuits*[3] de poids minimum du graphe des préfixes (une couverture par circuits est un ensemble de circuits disjoints couvrant tous les sommets). Le circuit $1 \to 2 \to \ldots \to n \to 1$ étant une couverture par circuits, (7.1) donne que le poids d'une couverture par circuits optimale est bien un minorant de OPT.

Contrairement au TSP optimal[4], on peut calculer une couverture par circuits optimale en temps polynomial. Associons au graphe des préfixes le graphe biparti $H = (U, V, U \times V)$ suivant. Les deux côtés de la partition comptent n sommets : $U = \{u_1, \ldots, u_n\}$ et $V = \{v_1, \ldots, v_n\}$. Pour tout $i, j \in \{1, \ldots, n\}$, l'arête $u_i v_j$ a pour poids $|\,\text{préfixe}(s_i, s_j)|$. Il est facile de voir qu'à toute couverture par circuits du graphe des préfixes correspond un couplage parfait de même poids dans H et réciproquement. Par conséquent, trouver une couverture par circuits optimale se ramène à trouver un couplage parfait de poids minimum dans H.

Pour tout circuit $c = (i_1 \to i_2 \to \cdots \to i_\ell \to i_1)$ du graphe des préfixes, posons

$$\alpha(c) = \text{préfixe}(s_{i_1}, s_{i_2}) \ldots \text{préfixe}(s_{i_{l-1}}, s_{i_\ell}).\,\text{préfixe}(s_{i_\ell}, s_{i_1}).$$

$\text{poids}(c) = |\alpha(c)|$ est le *poids du circuit* c. Remarquons que tous les mots $s_{i_1}, s_{i_2}, \ldots, s_{i_\ell}$ sont facteurs de $(\alpha(c))^\infty$. Posons ensuite

$$\sigma(c) = \alpha(c).s_{i_1}.$$

[3] *Cycle cover*, en anglais.

[4] TSP (traveling salesman problem) : problème du voyageur de commerce.

$\sigma(c)$ est un surfacteur[5] de $s_{i_1}, \ldots, s_{i_\ell}$.Dans la construction précédente, nous avons « ouvert » le circuit c en un mot arbitraire s_{i_1}. Nous dirons que s_{i_1} est le *mot représentatif* de c. Voici l'algorithme :

Algorithme 7.1 (Surfacteur minimum – 4-approximation)

1. Construire le graphe des préfixes associés aux mots de S.
2. Trouver une couverture par circuits $\mathcal{C} = \{c_1, \ldots, c_k\}$ de poids minimum du graphe des préfixes.
3. Renvoyer le mot $\sigma(c_1) \ldots \sigma(c_k)$.

La sortie de l'algorithme est clairement un surfacteur des mots de S. Remarquez que si l'on pouvait trouver pour chaque circuit un mot représentatif de longueur inférieure au poids du circuit, alors le mot renvoyé serait de longueur $\leqslant 2 \cdot \mathrm{OPT}$ (équation (7.1)). Le cas difficile est donc celui où il existe au moins un circuit dont tous les mots représentatifs sont longs. Dans ce cas néanmoins, le lemme 7.2 démontre que ces mots représentatifs sont en quelque sorte « périodiques » car ils sont tous facteurs de $(\alpha(c))^\infty$. Nous utiliserons ce fait pour prouver le lemme 7.3 qui conduit à un autre minorant de OPT.

Lemme 7.2 *Soit $S' \subseteq S$. S'il existe un mot t tel que tout mot de S' est un facteur de t^∞, alors il existe dans le graphe des préfixes un circuit de poids inférieur à $|t|$ qui couvre exactement les sommets correspondants aux mots de S'.*

Preuve : Regardons les positions des premières occurrences de chaque mot de S' dans t^∞. Ces positions sont toutes distinctes (car aucun mot de S n'est facteur d'un autre) et appartiennent à la première copie de t. Considérons le circuit du graphe des préfixes, qui parcourt les sommets correspondants dans l'ordre donné par ces positions. Son poids est clairement inférieur à $|t|$. □

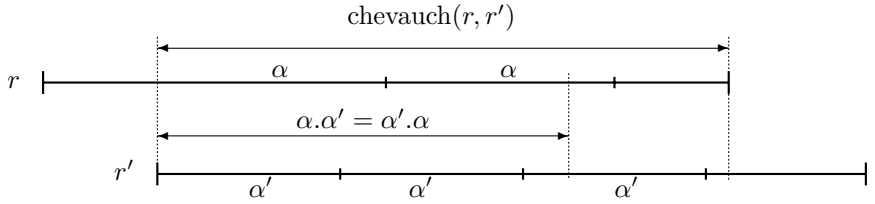
Lemme 7.3 *Soient c et c' deux circuits de la couverture par circuits $\mathcal{C}$, et r et r' des mots représentatifs de ces circuits. Alors*

$$|\operatorname{chevauch}(r, r')| < \operatorname{poids}(c) + \operatorname{poids}(c').$$

Preuve : Supposons par l'absurde que $|\operatorname{chevauch}(r, r')| \geq \operatorname{poids}(c) + \operatorname{poids}(c')$. Notons α et α' les préfixes de $\operatorname{chevauch}(r, r')$ de

[5] C'est aussi vrai pour le mot plus court $\alpha(c).\operatorname{chevauch}(s_\ell, s_1)$. Nous travaillerons avec $\sigma(c)$, car il nous sera nécessaire pour la 3-approximation de la section suivante, où nous utiliserons que $\sigma(c)$ commence et finit par une copie de s_{i_1}.

longueur poids(c) et poids(c'), respectivement.



Remarquons que chevauch(r, r') est un préfixe de α^∞ et de ${\alpha'}^\infty$. D'autre part, α est un préfixe de ${\alpha'}^\infty$ et α' de α^∞. Or, chevauch$(r, r') \geq |\alpha| + |\alpha'|$. Ainsi, α et α' commutent, c'est-à-dire $\alpha.\alpha' = \alpha'.\alpha$. Par conséquent, $\alpha^\infty = {\alpha'}^\infty$, car pour tout $k > 0$, $\alpha^k.{\alpha'}^k = {\alpha'}^k.\alpha^k$, et donc tout préfixe de α^∞ est aussi préfixe de ${\alpha'}^\infty$.

D'après le lemme 7.2, il existe donc dans le graphe des préfixes un circuit de poids inférieur à $|\alpha| = \text{poids}(c)$, qui couvre tous les mots de c et de c', ce qui contredit la minimalité de la couverture par circuits $\mathcal{C}$, CQFD. □

Théorème 7.4 *L'algorithme 7.1 est une 4-approximation pour le problème du surfacteur minimum.*

Preuve : Posons $\text{poids}(\mathcal{C}) = \sum_{i=1}^{k} \text{poids}(c_i)$. La longueur de la sortie de l'algorithme est

$$\sum_{i=1}^{k} |\sigma(c_i)| = \text{poids}(\mathcal{C}) + \sum_{i=1}^{k} |r_i|,$$

où r_i est le mot représentatif du circuit c_i. Nous avons prouvé que $\text{poids}(\mathcal{C}) \leq \text{OPT}$. Reste à démontrer que la somme des longueurs des mots représentatifs est inférieure à $3 \cdot \text{OPT}$.

Supposons que $r_1, \ldots, r_k$ sont numérotés par ordre de leur première occurrence à gauche dans le surfacteur optimal de S. Le lemme 7.3 donne le minorant suivant pour OPT :

$$\text{OPT} \geqslant \sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |\,\text{chevauch}(r_i, r_{i+1})| \geqslant \sum_{i=1}^{k} |r_i| - 2\sum_{i=1}^{k} \text{poids}(c_i).$$

Par conséquent,

$$\sum_{i=1}^{k} |r_i| \leqslant \text{OPT} + 2\sum_{i=1}^{k} \text{poids}(c_i) \leqslant 3 \cdot \text{OPT}\,.$$

□

7.2 Réduction à 3 du facteur d'approximation

Remarquons que tout surfacteur des mots $(\sigma(c_i))_{i=1,\ldots,k}$ est également un surfacteur des mots de S. Aussi, au lieu de simplement concaténer ces mots, recouvrons-les les uns les autres le plus possible (malgré les apparences, nous ne tournons pas en rond, comme nous allons le démontrer maintenant !).

Pour tout ensemble de mots X, notons $||X||$ la somme des longueurs des mots de X. Soit s un surfacteur de S. La *compression* réalisée par s est définie comme la différence entre la somme des longueurs des mots de S et la longueur de s, c'est-à-dire $||S|| - |s|$. La compression maximale est obtenue par le surfacteur minimum. Il existe différents algorithmes qui réalisent au moins la moitié de la compression optimale. Un tel exemple est l'algorithme glouton pour le surfacteur décrit section 2.3. Cependant, l'évaluation des performances de cet algorithme repose sur une analyse complexe de cas. Un algorithme moins performant sera présenté section 7.2.1. Ces deux algorithmes conviennent parfaitement pour la troisième phase de l'algorithme 7.5.

Algorithme 7.5 (Surfacteur minimum – 3-approximation)

1. Construire le graphe des préfixes des mots de S.
2. Trouver un couverture par circuits optimale $\mathcal{C} = \{c_1, \ldots, c_k\}$ du graphe des préfixes.
3. Lancer l'algorithme glouton pour le surfacteur sur $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ et renvoyer le résultat τ.

Notons OPT_σ la longueur du surfacteur minimum de $S_\sigma = \{\sigma(c_1), \ldots, \sigma(c_k)\}$, et prenons un mot représentatif r_i pour chaque c_i.

Lemme 7.6 $\quad |\tau| \leqslant \mathrm{OPT}_\sigma + \mathrm{poids}(\mathcal{C})$.

Preuve : Supposons sans perte de généralité que $\sigma(c_1)$,...,$\sigma(c_k)$ apparaissent dans cet ordre dans un surfacteur minimum de S_σ. La compression optimale pour S_σ est donnée par

$$\sum_{i=1}^{k-1} |\,\mathrm{chevauch}(\sigma(c_i), \sigma(c_{i+1}))|.$$

Comme r_i est un préfixe et un suffixe de chaque $\sigma(c_i)$ et qu'aucun r_i n'est facteur d'un autre, le lemme 7.3 donne

$$|\,\mathrm{chevauch}(\sigma(c_i), \sigma(c_{i+1}))| \leq \mathrm{poids}(c_i) + \mathrm{poids}(c_{i+1}).$$

Par conséquent, la compression optimale sur S_σ est bornée par $2 \cdot \mathrm{poids}(\mathcal{C})$, c'est-à-dire $||S_\sigma|| - \mathrm{OPT}_\sigma \leq 2 \cdot \mathrm{poids}(\mathcal{C})$.

Or la compression obtenue par l'algorithme glouton pour le surfacteur sur S_σ est d'au moins la moitié de la compression maximale. Ainsi,

$$||S_\sigma|| - |\tau| \geqslant \frac{1}{2}(||S_\sigma|| - \mathrm{OPT}_\sigma).$$

Et donc,

$$2(|\tau| - \mathrm{OPT}_\sigma) \leqslant ||S_\sigma|| - \mathrm{OPT}_\sigma \leq 2 \cdot \mathrm{poids}(\mathcal{C}),$$

ce qui conclut la preuve du lemme. □

Il nous reste à relier OPT_σ avec OPT.

Lemme 7.7 $\mathrm{OPT}_\sigma \leqslant \mathrm{OPT} + \mathrm{poids}(\mathcal{C})$.

Preuve : Notons OPT_r la longueur d'un surfacteur minimum des mots de $S_r = \{r_1, \ldots, r_k\}$. La clé est que r_i est à la fois préfixe et suffixe de $\sigma(c_i)$. Par conséquent, la compression optimale sur S_σ est au moins aussi grande que celle sur S_r, c'est-à-dire

$$||S_\sigma|| - \mathrm{OPT}_\sigma \geqslant ||S_r|| - \mathrm{OPT}_r\,.$$

Or, clairement, $||S_\sigma|| = ||S_r|| + \mathrm{poids}(\mathcal{C})$ et donc,

$$\mathrm{OPT}_\sigma \leqslant \mathrm{OPT}_r + \mathrm{poids}(\mathcal{C}).$$

Enfin $\mathrm{OPT}_r \leqslant \mathrm{OPT}$, CQFD. □

Les deux lemmes précédents donnent finalement :

Théorème 7.8 *L'algorithme 7.5 est une 3-approximation pour le problème du surfacteur minimum.*

7.2.1 Obtention de la moitié de la compression optimale

Nous présentons ici un algorithme qui réalise au moins la moitié de la compression optimale. Imaginons que les mots à compresser $s_1, \ldots, s_n$ sont numérotés dans leur ordre de première occurrence à gauche dans un surfacteur minimum. Alors la compression optimale vaut

$$\sum_{i=1}^{n-1} |\,\mathrm{chevauch}(s_i, s_{i+1})|.$$

C'est le poids du circuit du voyageur de commerce $1 \to 2 \to \ldots \to n$ dans le *graphe de chevauchement* H des mots $s_1, \ldots, s_n$. Le graphe orienté H est défini comme suit : H a un sommet v_i pour chaque mot s_i, et un arc

$(v_i \to v_j)$ de poids $|\,\text{chevauch}(s_i, s_j)|$ pour tout $i \neq j$, $1 \leqslant i, j \leqslant n$ (H n'a pas de boucle).

La compression optimale est majorée par le coût maximal d'un circuit de voyageur de commerce dans H, lui-même majoré par le coût d'une couverture par circuits maximum. Or, cette dernière se calcule en temps polynomial en utilisant un algorithme de couplage, comme nous l'avons vu pour la couverture par circuits minimum. Tout circuit de H est de longueur au moins 2, car H n'a pas de boucle. Ôtons l'arc le moins lourd de chacun des circuits d'une couverture maximum ; nous obtenons un ensemble de chemins disjoints. Recouvrons les mots $s_1, \ldots, s_n$ en suivant les arcs de ces chemins et concaténons les mots obtenus. Nous construisons ainsi un surfacteur réalisant au moins la moitié de la compression optimale.

7.3 Exercices

7.1 Démontrez qu'on ne peut pas remplacer l'inégalité du lemme 7.3 par

$$|\,\text{chevauch}(r, r')| < \max\{\text{poids}(c), \text{poids}(c')\}.$$

7.2 (Jiang, Li, et Du [156]) Donner une approximation à un facteur constant pour les variantes du problème du surfacteur minimum proposées dans l'exercice 2.16.

7.4 Notes

Les algorithmes présentés dans ce chapitre sont dus à Blum, Jiang, Li, Tromp, et Yannakakis [29].

8 Sac à dos

Nous avons mentionné au chapitre 1 l'existence de problèmes d'optimisation **NP**-difficiles que l'on peut approcher avec une précision arbitraire. Dans ce chapitre, nous formalisons cette notion et démontrons que le problème du sac à dos admet un tel schéma d'approximation.

Considérons un problème d'optimisation **NP**-difficile Π, de fonction objectif f_Π. Un algorithme $\mathcal{A}$ est un *schéma d'approximation*[1] pour Π si pour toute entrée (I, ε), où I est une instance de Π et $\varepsilon > 0$ la précision demandée, la solution renvoyée s vérifie :

- $f_\Pi(I, s) \leqslant (1 + \varepsilon) \cdot \mathrm{OPT}$, si Π est un problème de minimisation.
- $f_\Pi(I, s) \geqslant (1 - \varepsilon) \cdot \mathrm{OPT}$, si Π est un problème de maximisation.

$\mathcal{A}$ est un *schéma d'approximation polynomial*,[2] PTAS en abrégé, si pour tout $\varepsilon > 0$ *fixé*, le temps de calcul de $\mathcal{A}$ est borné par un polynôme en la taille de l'instance I.

La définition ci-dessus permet une dépendance arbitraire du temps de calcul de $\mathcal{A}$ en fonction de ε. La définition suivante propose un cadre plus restrictif. $\mathcal{A}$ est un *schéma d'approximation totalement polynomial*,[3] FPTAS en abrégé, si c'est un schéma d'approximation polynomial dont le temps de calcul est borné par un polynôme en la taille de l'instance I et en $1/\varepsilon$, pour tout (I, ε).

Techniquement, un FPTAS est ce qu'on peut espérer de mieux pour un problème d'optimisation **NP**-difficile, à moins que $\mathbf{P} = \mathbf{NP}$; référez-vous à la section 8.3.1 pour une discussion à ce propos. Le problème du sac à dos admet un FPTAS.

Problème 8.1 (Sac à dos)[4] Étant donné un ensemble d'objets $S = \{a_1, \ldots, a_n\}$, définis par leurs tailles et leurs valeurs, taille$(a_i) \in \mathbf{N}$ et valeur$(a_i) \in \mathbf{N}$, et la « capacité du sac à dos » $B \in \mathbf{N}$, déterminer un sous-ensemble d'objets dont la taille totale (le volume) est inférieure à B et dont la valeur totale (le gain) soit maximum.

Un algorithme évident pour ce problème consiste à trier les objets par ratio valeur/taille décroissant, puis à sélectionner gloutonnement les objets

[1] *Approximation scheme*, en anglais.
[2] *Polynomial time approximation scheme (PTAS)*, en anglais.
[3] *Fully polynomial time approximation scheme (FPTAS)*, en anglais.
[4] *Knapsack problem*, en anglais.

dans cet ordre. Il est néanmoins facile de montrer que cet algorithme peut produire des résultats arbitrairement mauvais (voir exercice 8.1).

8.1 Un algorithme pseudo-polynomial pour le sac à dos

Avant de décrire un FPTAS pour le sac à dos, nous avons besoin d'un concept supplémentaire. Une instance d'un problème d'optimisation Π est une suite d'*objets* (des ensembles, des graphes...) et de *nombres* (des coûts, valeurs, tailles...). Jusqu'à présent, nous avons considéré que tous les nombres présents dans chaque instance I étaient codés en binaire. La *taille* de l'instance, notée $|I|$, était définie comme le nombre de bits nécessaires pour écrire I avec cette convention. Notons I_u l'instance I où tous les nombres sont écrits en unaire. La *taille en unaire* de l'instance I, notée $|I_u|$, est alors le nombre de bits nécessaires pour écrire I_u.

Usuellement, un algorithme est dit efficace pour un problème Π si son temps de calcul sur toute instance I est borné par un polynôme en $|I|$. Voici une notion plus faible d'efficacité. Un algorithme pour un problème Π est dit *pseudo-polynomial*[5] si son temps de calcul est borné par un polynôme en $|I_u|$ pour toute instance I.

Étant **NP**-difficile, le problème du sac à dos n'admet pas d'algorithme polynomial ; il admet cependant un algorithme pseudo-polynomial. Ce fait est critique pour démontrer l'existence d'un FPTAS pour le sac à dos. Tous les algorithmes pseudo-polynomiaux connus utilisent la programmation dynamique.

Notons P la valeur maximale d'un objet de S, $P = \max_{a \in S} \text{valeur}(a)$. Alors $n \cdot P$ est une majoration triviale du gain optimal pour le sac à dos. Pour tout $i \in \{1, \ldots, n\}$ et $p \in \{1, \ldots, nP\}$, notons $S_{i,p}$ un sous-ensemble de $\{a_1, \ldots, a_i\}$ de gain exactement p et dont le volume est minimum. Notons $A(i,p)$ le volume de $S_{i,p}$ ($A(i,p) = \infty$ s'il n'existe pas de tel ensemble). $A(1,p)$ est trivial à calculer pour tout $p \in \{1, \ldots, nP\}$. La récurrence suivante permet de calculer tous les $A(i,p)$ en temps $O(n^2P)$:

$$A(i+1,p) = \begin{cases} \min\{A(i,p),\ \text{taille}(a_{i+1}) + A(i, p - \text{valeur}(a_{i+1}))\} & \text{si valeur}(a_{i+1}) \leqslant p \\ A(i+1,p) = A(i,p) & \text{sinon} \end{cases}$$

Le gain maximum est donc donné par $\max\{p : A(n,p) \leq B\}$ pour un sac à dos de taille B. Cet algorithme résout bien en temps pseudo-polynomial le problème du sac à dos.

[5] *Pseudo-polynomial time algorithm*, en anglais.

8.2 Un FPTAS pour le sac à dos

Remarquons que si les valeurs des objets étaient des petits nombres, c'est-à-dire bornés par un polynôme en n, alors l'algorithme précédent serait polynomial puisque son temps de calcul serait borné par un polynôme en $|I|$. La clé pour obtenir un FPTAS est d'exploiter ce fait : nous allons ignorer un certain nombre des bits les moins significatifs des valeurs des objets (ce nombre dépendra de la précision ε), de sorte que ces valeurs « arrondies » puissent être considérées comme des nombres bornés par un polynôme en n et $1/\varepsilon$. Nous obtiendrons ainsi, en un temps borné par n et $1/\varepsilon$, une solution de gain au moins égal à $(1-\varepsilon)\cdot \mathrm{OPT}$.

Algorithme 8.2 (FPTAS pour le sac à dos)

1. Étant donné $\varepsilon > 0$, poser $K = \frac{\varepsilon P}{n}$.
2. Pour chaque objet a_i, poser $\mathsf{valeur}'(a_i) = \left\lfloor \frac{\mathsf{valeur}(a_i)}{K} \right\rfloor$.
3. Lancer l'algorithme de programmation dynamique avec ces valeurs arrondies et trouver un ensemble S' de gain (arrondi) maximum.
4. Renvoyer S'.

Lemme 8.3 *L'ensemble renvoyé par l'algorithme 8.2 vérifie*

$$\mathrm{valeur}(S') \geq (1-\varepsilon)\cdot \mathrm{OPT}\,.$$

Preuve : Notons O un ensemble de gain (non arrondi) maximum. Pour tout objet a, du fait de l'arrondi vers le bas, $K \cdot \mathrm{valeur}'(a)$ est inférieur ou égal à $\mathrm{valeur}(a)$, mais pas de plus que K. Ainsi,

$$\mathrm{valeur}(O) - K \cdot \mathrm{valeur}'(O) \leqslant nK.$$

L'étape de programmation dynamique renvoie un ensemble de gain au moins aussi bon que celui de O avec les valeurs arrondies. Par conséquent,

$$\begin{aligned}\mathrm{valeur}(S') &\geqslant K \cdot \mathrm{valeur}'(O) \geqslant \mathrm{valeur}(O) - nK = \mathrm{OPT} - \varepsilon P \\ &\geqslant (1-\varepsilon)\cdot \mathrm{OPT}\,,\end{aligned}$$

en remarquant que $\mathrm{OPT} \geqslant P$ pour obtenir la dernière inégalité. □

Théorème 8.4 *L'algorithme 8.2 est un schéma d'approximation totalement polynomial pour le problème du sac à dos.*

Preuve : D'après le lemme 8.3, la solution renvoyée a un gain supérieur à $(1-\varepsilon)\cdot\mathrm{OPT}$. Le temps de calcul de l'algorithme est $O\left(n^2 \left\lfloor \frac{P}{K} \right\rfloor\right) = O\left(n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor\right)$. C'est bien un polynôme en n et $1/\varepsilon$, CQFD. □

8.3 Existence de FPTAS et NP-difficulté forte

Nous allons démontrer dans cette section que seul un très petit nombre des problèmes **NP**-difficiles connus admettent un FTPAS. Tout d'abord, voici une version forte de la **NP**-difficulté, en liaison étroite avec la notion d'algorithme pseudo-polynomial, qui est une version faible de la notion d'algorithme efficace. Un problème Π est **NP**-*difficile au sens fort* si pour tout problème Π' de **NP**, il existe une réduction polynomiale de Π' à Π telle que les nombres soient tous écrits en unaire dans le problème réduit (Π).

Cette condition impose en particulier que la réduction n'utilise que des nombres bornés par un polynôme en la taille de l'instance. La plupart des problèmes **NP**-difficiles le sont au sens fort ; c'est le cas par exemple, de tous les problèmes des chapitres précédents pour lesquels nous avons décrit des algorithmes d'approximation. Un problème **NP**-difficile au sens fort n'admet pas d'algorithme pseudo-polynomial, à moins que $\mathbf{P} = \mathbf{NP}$ (voir exercice 8.4). Par conséquent, le sac à dos n'est pas **NP**-difficile au sens fort, à moins que $\mathbf{P} = \mathbf{NP}$.

Nous allons maintenant démontrer que sous certaines conditions très faibles, tout problème **NP**-difficile qui admet un FPTAS, admet aussi un algorithme pseudo-polynomial. Nous démontrons ce théorème dans le cadre d'un problème de minimisation, la preuve est similaire pour un problème de maximisation.

Théorème 8.5 *Soient p un polynôme et Π un problème* **NP**-*difficile de minimisation tel que sa fonction objectif f_Π ne prenne que des valeurs entières et tel que pour toute instance I, $\mathrm{OPT}(I) < p(|I_u|)$. Si Π admet un FPTAS, alors il admet aussi un algorithme pseudo-polynomial.*

Preuve : Considérons un FPTAS pour Π dont le temps de calcul est borné par un polynôme $q(|I|, 1/\varepsilon)$ sur toute instance I et pour toute précision ε.

Pour toute instance I, nous fixons la précision à $\varepsilon = 1/p(|I_u|)$, et lançons le FPTAS. La solution produite a une valeur objectif inférieure ou égale à :

$$(1+\varepsilon)\,\mathrm{OPT}(I) < \mathrm{OPT}(I) + \varepsilon p(|I_u|) = \mathrm{OPT}(I) + 1.$$

Le FTPAS est donc contraint à produire une solution optimale pour cette précision. Le temps de calcul est borné par $q(|I|, p(|I_u|))$, c'est-à-dire par un polynôme en $|I_u|$. Nous avons donc un algorithme pseudo-polynomial pour Π. □

Le corollaire suivant s'applique à la plupart des problèmes **NP**-difficiles connus.

Corollaire 8.6 *Soit Π un problème d'optimisation* **NP***-difficile vérifiant les hypothèses du théorème 8.5. Si Π est* **NP***-difficile au sens fort, alors Π n'admet pas de FPTAS, à moins que* $\mathbf{P} = \mathbf{NP}$.

Preuve : D'après le théorème 8.5, si Π admet un FPTAS, alors il admet un algorithme pseudo-polynomial, et ne peut donc être **NP**-difficile au sens fort, à moins que $\mathbf{P} = \mathbf{NP}$. □

Remplacer l'hypothèse sur OPT du théorème 8.5 par $\mathrm{OPT}(I) < p(|I|)$ impliquerait qu'il existe un algorithme polynomial pour Π. Cette condition plus restrictive a bien moins d'applications. Par exemple, le problème de l'ordonnancement de temps total d'exécution minimum étudié au chapitre 10 ne la vérifie pas.

8.3.1 Existe-t-il de meilleures approximations qu'un FPTAS ?

La plupart des FTPAS et PTAS connus reposent sur un compromis entre précision et temps de calcul – l'instance du problème est transformée en une instance plus grossière, dépendant de la précision ε souhaitée, qui est résolue de manière optimale par programmation dynamique. Cette résolution optimale est en fait une recherche exhaustive dans un domaine de taille polynomiale : pour le sac à dos, il s'agit de calculer $A(i, p)$ pour tous les (i, p). Pour la plupart de ces algorithmes, le temps de calcul est prohibitif, même pour des valeurs raisonnables de n et ε. De plus, puisqu'un tel algorithme recourt à une recherche exhaustive, est-il réellement en mesure de produire une solution efficacement ? Est-ce qu'un FTPAS ou un PTAS sont les meilleures solutions pour un problème **NP**-difficile ? Cette question est sans aucun doute complexe et il n'existe pas réponse évidente.

8.4 Exercices

8.1 Étudions l'algorithme glouton suivant pour le sac à dos : trier les objets par ratio valeur/taille décroissant, puis sélectionner les objets dans cet ordre de manière gloutonne. Démontrez que cet algorithme peut produire des solutions arbitrairement mauvaises.

8.2 Étudions la variante suivante de l'algorithme décrit dans l'exercice 8.1. Notons $a_1, \ldots, a_n$ les objets triés. Déterminer le plus petit k tel que le volume des k premiers objets soit supérieur (ou égal) à une valeur donnée B. Puis,

sélectionner le meilleur des deux ensembles $\{a_1, \ldots, a_{k-1}\}$ et $\{a_k\}$ (nous supposons que tous les objets sont de taille inférieure à B). Démontrez que cet algorithme est une 2-approximation.

8.3 (Bazgan, Santha, et Tuza [24]) Exhibez un FPTAS pour le problème suivant.

Problème 8.7 (Ensembles somme-ratio minimaux)[6] Étant donné n entiers positifs, $0 < a_1 < \ldots < a_n$, déterminer deux sous-ensembles disjoints non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ tels que $\sum_{i \in S_1} a_i \geqslant \sum_{i \in S_2} a_i$, et tels que le ratio

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i}$$

soit minimal.
Indication : Commencez par exhiber un algorithme pseudo-polynomial pour ce problème. Puis arrondissez les entrées convenablement.

8.4 Démontrez qu'aucun problème **NP**-difficile au sens fort n'admet d'algorithme pseudo-polynomial, à moins que $\mathbf{P} = \mathbf{NP}$.

8.5 Notes

L'algorithme 8.2 est dû à Ibarra et Kim [142]. Le théorème 8.5 a été démontré par Garey et Johnson [99].

[6] *Subset-sum ratio problem*, en anglais.

9 Empaquetage

Nous étudions ici le problème suivant.

Problème 9.1 (Empaquetage)[1] Étant donné n objets de tailles $a_1, \ldots, a_n$ dans $]0, 1]$, ranger les objets dans un nombre minimum de boîtes de taille unitaire.

Ce problème a de très nombreuses applications industrielles. Par exemple, la découpe de matériau : la taille des boîtes correspond à la taille standard du matériau, et les objets aux longueurs commandées par les clients.

Il est facile d'obtenir une 2-approximation pour ce problème. Considérons, par exemple, l'algorithme suivant, appelé First-Fit en anglais[2]. Cet algorithme glouton considère les objets dans un ordre arbitraire. Au i-ième pas, il a déjà rempli un certain nombre de boîtes $B_1, \ldots, B_k$, numérotées par ordre de première ouverture. First-Fit parcourt les boîtes dans cet ordre et place l'objet courant, a_i, dans la première qui convient ; si a_i ne tient dans aucune de ces boîtes, il en ouvre une nouvelle B_{k+1} et y place a_i. Lorsque cet algorithme ouvre m boîtes, les $m - 1$ premières boîtes sont au moins à moitié pleines. Par conséquent,

$$\sum_{i=1}^{n} a_i > \frac{m-1}{2}.$$

Or la somme des tailles des objets est un minorant de OPT et donc $m - 1 < 2\,\mathrm{OPT}$, c'est-à-dire $m \leqslant 2\,\mathrm{OPT}$ (se reporter aux notes pour une meilleure analyse des performances de cet algorithme). Voici un résultat d'impossibilité :

Théorème 9.2 *Pour tout $\varepsilon > 0$, il n'existe pas de $(3/2 - \varepsilon)$-approximation pour le problème de l'empaquetage, à moins que* $\mathbf{P} = \mathbf{NP}$.

Preuve : Démontrons par l'absurde que si tel était le cas, alors il existerait une solution polynomiale pour le problème **NP**-difficile de la partition d'un ensemble de n entiers strictement positifs $a_1, \ldots, a_n$ en deux ensembles de somme égale à $\frac{1}{2}\sum_i a_i$. Clairement, une telle partition existe ssi les n entiers

[1] *Bin packing*, en anglais.

[2] Littéralement, « la-première-boîte-qui-convient ».

peuvent être empaquetés dans deux boîtes de taille $\frac{1}{2}\sum_i a_i$. Et si tel est le cas, une $(3/2-\varepsilon)$-approximation pour l'empaquetage proposerait un empaquetage optimal, et donc résoudrait le problème de partition. □

9.1 Un PTAS asymptotique

Remarquons que le résultat d'impossibilité du théorème 9.2 repose sur des instances très particulières, où OPT est borné par un petit nombre (2 ou 3) quel que soit le nombre des objets. Que peut-on dire d'instances plus « typiques », où OPT croît avec n ? Voici une réponse :

Théorème 9.3 *Pour tout $0 < \varepsilon \leqslant 1/2$, il existe un algorithme $\mathcal{A}_\varepsilon$ polynomial en n qui calcule un empaquetage dans moins de $(1+2\varepsilon)\,\mathrm{OPT}+1$ boîtes.*

La suite d'algorithmes $(\mathcal{A}_\varepsilon)$ est un *schéma d'approximation polynomial asymptotique*[3] pour l'empaquetage, car pour tout $\varepsilon > 0$, il existe un entier $N > 0$, et un algorithme polynomial $\mathcal{B}_N$ dans cette suite, tels que $\mathcal{B}_N$ soit une $(1+\varepsilon)$-approximation pour toutes les instances telles que $\mathrm{OPT} \geqslant N$. On ne peut pas cependant considérer que le théorème 9.3 soit une solution raisonnable en pratique car le temps de calcul des algorithmes $\mathcal{A}_\varepsilon$ est très élevé.

Nous allons démontrer le théorème 9.3 en trois étapes.

Lemme 9.4 *Soient $\varepsilon > 0$ et K un entier positif, fixés. Il existe un algorithme polynomial pour résoudre la restriction du problème d'empaquetage aux instances où les objets sont de taille au moins égale à ε et où le nombre de tailles possibles est inférieur à K.*

Preuve : Le nombre d'objets dans chaque boîte est majoré par $\lfloor 1/\varepsilon \rfloor$. Notons M cette quantité. Le nombre de possibilités pour remplir les boîtes est donc borné par $R = \binom{M+K}{M}$ (voir exercice 9.4). R est une constante (grande !). Notons qu'au plus n boîtes seront utilisées. Ainsi, le nombre d'empaquetages possibles est borné par $P = \binom{n+R}{R}$. P est un polynôme en n (voir exercice 9.4). Énumérons tous ces empaquetages et sélectionnons le meilleur, nous obtenons une solution optimale. □

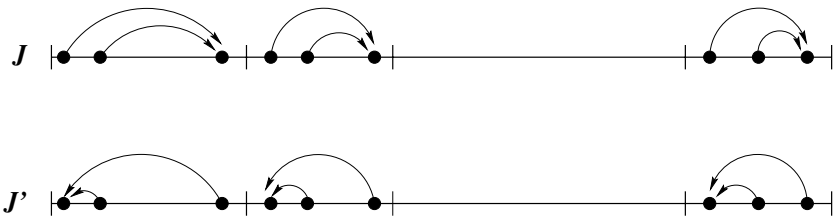
Lemme 9.5 *Soit $\varepsilon > 0$ fixé. Il existe une $(1+\varepsilon)$-approximation pour la restriction du problème d'empaquetage aux instances où tous les objets sont de taille au moins ε.*

Preuve : Notons I une telle instance. Trions les n objets par taille croissante et partitionnons-les dans cet ordre en $K = \lceil 1/\varepsilon^2 \rceil$ groupes comptant au

[3] *Asymptotic polynomial time approximation scheme (APTAS)*, en anglais.

plus $Q = \lfloor n\varepsilon^2 \rfloor$ objets chacun. Remarquons que deux objets de même taille peuvent être mis dans deux groupes différents.

Nous construisons une nouvelle instance J en arrondissant les tailles de chaque objet à la taille du plus grand objet de son groupe. Les tailles des objets de l'instance J prennent donc K valeurs différentes au plus. Par conséquent, d'après le lemme 9.4, nous pouvons trouver un empaquetage optimal pour J. Clairement, cet empaquetage est un empaquetage valide des objets avec leurs tailles réelles. Nous allons maintenant montrer que le nombre de boîtes utilisées $\mathrm{OPT}(J)$ vérifie : $\mathrm{OPT}(J) \leqslant (1+\varepsilon)\,\mathrm{OPT}(I)$, ce qui conclura la preuve du lemme.



Un argument astucieux permet d'obtenir ce résultat. Construisons une autre instance J' en arrondissant cette fois-ci la taille réelle de chaque objet à celle du plus petit objet de son groupe. Clairement $\mathrm{OPT}(J') \leqslant \mathrm{OPT}(I)$. La clé est que tout empaquetage de J' est un empaquetage valide de tous les objets de J sauf pour les Q objets les plus grands (c'est-à-dire, ceux du dernier groupe) — l'exercice 9.6 propose de démontrer ce fait. Par conséquent,

$$\mathrm{OPT}(J) \leqslant \mathrm{OPT}(J') + Q \leqslant \mathrm{OPT}(I) + Q.$$

Or la taille de chaque objet de I est supérieure à ε, donc $\mathrm{OPT}(I) \geqslant n\varepsilon$. Ainsi, $Q = \lfloor n\varepsilon^2 \rfloor \leqslant \varepsilon\,\mathrm{OPT}$. D'où, $\mathrm{OPT}(J) \leqslant (1+\varepsilon)\,\mathrm{OPT}(I)$. □

Preuve du théorème 9.3 : Soit I l'instance à résoudre. Notons I' l'instance privées des objets de taille $< \varepsilon$ de I. D'après le lemme 9.5, nous pouvons calculer un empaquetage de I' dans moins de $(1+\varepsilon)\,\mathrm{OPT}(I')$ boîtes. Ensuite, empaquetons les petits objets restants (de taille $< \varepsilon$) en lançant l'algorithme First-Fit avec les boîtes ouvertes pour l'empaquetage de I'. Des boîtes supplémentaires sont ouvertes si aucune des boîtes déjà ouvertes ne convient.

Si aucune boîte supplémentaire n'est nécessaire, alors nous avons empaqueté les objets dans $(1+\varepsilon)\,\mathrm{OPT}(I') \leqslant (1+\varepsilon)\,\mathrm{OPT}(I)$ boîtes. Sinon, notons M le nombre total de boîtes utilisées. Clairement, toutes les boîtes, sauf la dernière, sont remplies à un niveau $> 1-\varepsilon$. Donc, la somme des tailles des objets de I est supérieure à $(M-1)(1-\varepsilon)$. Comme c'est aussi un minorant de OPT, nous obtenons (en utilisant l'hypothèse $\varepsilon \leqslant 1/2$) :

$$M \leqslant \frac{\mathrm{OPT}}{(1-\varepsilon)} + 1 \leqslant (1+2\varepsilon)\,\mathrm{OPT} + 1.$$

Finalement, pour tout $0 < \varepsilon \leqslant 1/2$, nous avons obtenu un algorithme garantissant d'utiliser moins de $(1+2\varepsilon)\,\mathrm{OPT}+1$ boîtes. □

Voici donc l'algorithme $\mathcal{A}_\varepsilon$.

Algorithme 9.6 (Algorithme d'empaquetage $\mathcal{A}_\varepsilon$)

1. Retirer les objets de taille $< \varepsilon$.
2. Arrondir pour obtenir un nombre constant de tailles différentes d'objets (lemme 9.5).
3. Trouver l'empaquetage optimal pour cette instance (lemme 9.4).
4. Utiliser cet empaquetage pour les objets initiaux.
5. Empaqueter les objets de taille $< \varepsilon$ avec First-Fit.

9.2 Exercices

9.1 Exhibez une instance pour laquelle First-Fit produit un empaquetage dans plus de $5/3 \cdot \mathrm{OPT}$ boîtes.

9.2 (Johnson [157]) Étudions un algorithme plus restreint que First-Fit, appelé Next-Fit en anglais[4], et qui essaie d'empaqueter l'objet courant uniquement dans la dernière boîte ouverte. Si elle ne convient pas, Next-Fit ouvre une nouvelle boîte et y met l'objet. Démontrez que Next-Fit est une 2-approximation. Proposez une instance critique.

9.3 (C. Kenyon) Un algorithme d'empaquetage est dit *monotone* lorsque le nombre de boîtes utilisées pour empaqueter chaque sous-ensemble de l'ensemble des objets est inférieur (ou égal) à celui utilisé pour empaqueter la totalité des objets. Montrez que Next-Fit est monotone et que First-Fit ne l'est pas.

9.4 Démontrez les bornes sur R et P données au lemme 9.4.
Indication : Remarquez que le nombre de façons de placer n boules identiques dans k urnes est $\binom{n+k-1}{n}$.

9.5 Voici une autre tentative pour obtenir le lemme 9.5. Pour tout $r \geqslant 0$, nous arrondissons à $\min(\varepsilon(1+\varepsilon)^{r+1}, 1)$ les objets dont la taille est dans

[4] Littéralement, « dans-la-boîte-courante ».

l'intervalle $]\varepsilon(1+\varepsilon)^r, \varepsilon(1+\varepsilon)^{r+1}]$. Ceci conduit clairement à un nombre fini de tailles d'objets. Ce schéma d'arrondi permet-il de terminer la preuve ?
Indication : Étudiez le cas où de nombreux objets ont pour taille $1/2$, et où $1/2 \neq \varepsilon(1+\varepsilon)^r$ pour tout $r \geqslant 0$.

9.6 Démontrez le fait suivant énoncé dans la preuve du lemme 9.5 : « un empaquetage des objets de J' est un empaquetage valide de tous les objets de J privé de ses Q plus grands objets ».
Indication : Ôtez les Q plus grands objets de J et les Q plus petits de J', et établissez un lien entre ces instances.

9.7 Proposez une autre preuve du lemme 9.4 en utilisant le fait que la programmation linéaire en nombres entiers avec un nombre fixé de variables est dans **P** — remarquons que comme le temps de calcul pour résoudre un problème linéaire en nombres entiers est exorbitant, cette variante est tout aussi impraticable.

9.8 Démontrez que s'il existe un algorithme garantissant une solution de coût $\mathrm{OPT}(I) + \log^2(\mathrm{OPT}(I))$ pour l'empaquetage, alors il existe un schéma d'approximation totalement polynomial pour ce problème.

9.9 (C. Kenyon) Étudions le problème suivant.

Problème 9.7 (Débordement)[5] Étant donné n objets de tailles $a_1, \ldots, a_n$ dans $]0, 1]$, placer les objets dans un nombre maximum de boîtes unitaires, de façon que toutes les boîtes débordent (la somme des tailles des objets dans chaque boîte doit être supérieure à 1).

Proposez un PTAS asymptotique pour ce problème restreint aux instances où les tailles des objets sont supérieures à une constante $c > 0$ fixée.
Indication : Utilisez les mêmes idées que l'algorithme 9.6.

9.3 Notes

Le premier résultat non trivial sur l'empaquetage fut que First-Fit ouvrait au plus $(17/10)\mathrm{OPT}+3$ boîtes, démontré par Ullman [257]. Le PTAS asymptotique est dû à Fernandez de la Vega et Lueker [93]. Une amélioration de l'algorithme garantissant $\mathrm{OPT}(I) + \log^2(\mathrm{OPT}(I))$, fut proposée par Karmarkar et Karp [171]. Pour plus de résultats, reportez-vous à l'état de l'art de Coffman, Garey et Johnson [53]. Le résultat cité dans l'exercice 9.7, démontrant que la programmation linéaire en nombres entiers avec un nombre fixé de variables est dans **P**, est dû à Lenstra [193]. L'empaquetage en ligne a fait l'objet de nombreux travaux (voir par exemple Borodin et El-Yaniv [33]).

[5] *Bin covering*, en anglais.

10 Minimisation du temps d'exécution total

Voici un des problèmes centraux de la théorie de l'ordonnancement.

Problème 10.1 (Minimiser le temps d'exécution total)[1] Étant donné les durées d'exécution $p_1, p_2, \ldots, p_n$ de n tâches (indépendantes) et un entier m, ordonnancer les tâches sur m machines identiques de façon à minimiser le *temps d'exécution total*.[2]

Ce chapitre présente une 2-approximation puis un PTAS pour ce problème.

10.1 Une 2-approximation

L'algorithme est très simple : nous traitons les tâches l'une après l'autre dans un ordre arbitraire, et nous plaçons la tâche courante systématiquement sur la machine courante la moins chargée. Cet algorithme repose sur les deux minorants suivants du temps d'exécution optimal OPT :

1. le temps de fonctionnement moyen d'une machine, $\left(\sum_i p_i\right)/m$, et
2. le temps d'exécution de la tâche la plus longue.

Notons LB la combinaison de ces deux minorants[3] :

$$\mathrm{LB} = \max\left\{ \frac{1}{m} \sum_i p_i,\ \max_i \{p_i\} \right\}.$$

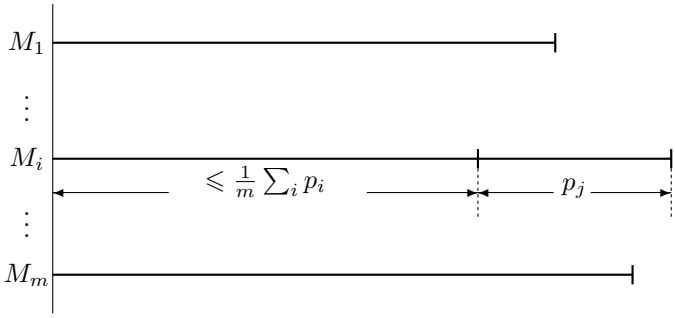
Algorithme 10.2 (Temps d'exécution total minimum)

1. Numéroter arbitrairement les tâches.
2. Placer les tâches sur les machines dans cet ordre, en choisissant pour chaque tâche la machine courante la moins chargée.

[1] *Minimum makespan scheduling*, en anglais.
[2] *Makespan* ou *completion time*, en anglais.
[3] *Lower bound*, en anglais.

Théorème 10.3 *L'algorithme 10.2 est une 2-approximation pour le problème de l'ordonnancement de temps d'exécution total minimum.*

Preuve : Soient M_i la machine la plus chargée dans l'ordonnancement calculé par l'algorithme et j l'indice de la dernière tâche exécutée sur cette machine.



Notons début$_j$ la date du début de l'exécution de la tâche j sur M_i. Puisque l'algorithme place la tâche sur la machine la moins chargée, toutes les machines sont occupées au-delà de début$_j$. Ainsi,

$$\text{début}_j \leqslant \frac{1}{m}\sum_i p_i \leqslant \text{OPT}\,.$$

Enfin, $p_j \leqslant \text{OPT}$. Le temps d'exécution total de cet ordonnancement vaut donc début$_j + p_j \leqslant 2$ OPT. □

Exemple 10.4 Voici une instance critique pour cet algorithme : considérons $n = m^2$ tâches unitaires suivies d'une tâche de durée d'exécution m. L'ordonnancement généré par l'algorithme exécute les tâches en temps $2m$, alors que $\text{OPT} = m + 1$. □

10.2 Un PTAS pour le temps d'exécution minimum

Le problème du temps d'exécution minimum est **NP**-difficile au sens fort ; le corollaire 8.6 implique que ce problème n'admet pas de FPTAS, à moins que $\mathbf{P} = \mathbf{NP}$. Nous pouvons cependant obtenir un PTAS pour ce problème en relevant des similitudes avec celui de l'empaquetage. En effet, un ordonnancement de temps d'exécution total t existe ssi il existe un empaquetage de n objets de tailles $p_1, p_2, \ldots, p_n$ dans m boîtes de taille t chacune. Cela conduit à la réduction suivante au problème de l'empaquetage. Notons I l'instance du problème de l'empaquetage avec n objets de tailles $p_1, \ldots, p_n$ et boîtes(I, t) le nombre minimum de boîtes de taille t utilisées pour les empaqueter. Le temps d'exécution optimal est donc :

$$\min\{t : \text{ boîtes}(I, t) \leqslant m\}.$$

Nous avons vu que le temps d'exécution minimum est minoré et majoré par LB et 2 LB respectivement. Nous pouvons donc le calculer par dichotomies dans cet intervalle. À première vue, cette réduction ne semble pas très utile car l'empaquetage est **NP**-difficile. Rappelons-nous cependant qu'il existe un algorithme polynomial pour l'empaquetage quand les tailles des objets appartiennent à un ensemble fini. Ce fait sera essentiel pour l'obtention de notre PTAS.

10.2.1 Empaquetage d'objets de tailles choisies dans un ensemble fini

Commençons par construire un algorithme fondé sur la programmation dynamique pour le problème d'empaquetage restreint. Cet algorithme améliore le lemme 9.4 de deux façons. Nous n'avons pas besoin de borner les tailles des objets et nous réduisons le temps de calcul. Soit k le nombre de tailles d'objets possibles ; supposons que toutes les boîtes ont pour taille 1. Toute instance du problème d'empaquetage est définie par le k-uplet $(i_1, i_2, \ldots, i_k)$, où i_j est le nombre d'objets de la j-ième taille possible. Notons $\text{BOÎTES}(i_1, i_2, \ldots, i_k)$ le nombre de boîtes utilisées par un empaquetage optimal des objets.

Considérons une instance $(n_1, n_2, \ldots, n_k)$ $(n = \sum_{i=1}^{k} n_i)$. Obtenons par énumération, l'ensemble $\mathcal{Q}$ de tous les k-uplets $(q_1, q_2, \ldots, q_k)$, $0 \leq q_i \leqslant n_i$ et $1 \leqslant i \leqslant k$, tels que $\text{BOÎTES}(q_1, q_2, \ldots, q_k) = 1$. $\mathcal{Q}$ a clairement $O(n^k)$ éléments. Puis, remplissons la table k-dimensionnelle $\text{BOÎTES}(i_1, i_2, \ldots, i_k)$, pour $(i_1, i_2, \ldots, i_k) \in \{0, \ldots, n_1\} \times \{0, \ldots, n_2\} \times \cdots \times \{0, \ldots, n_k\}$. Fixons $\text{BOÎTES}(q) = 1$ pour tous les $q \in \mathcal{Q}$, puis utilisons la récurrence suivante pour compléter la table :

$$\text{BOÎTES}(i_1, i_2, \ldots, i_k) = 1 + \min_{q \in \mathcal{Q}} \text{BOÎTES}(i_1 - q_1, \ldots, i_k - q_k). \qquad (10.1)$$

Le temps de calcul de chaque entrée dans la table est $O(n^k)$. Ainsi, le temps de calcul de la table (et donc de $\text{BOÎTES}(n_1, n_2, \ldots, n_k)$) vaut $O(n^{2k})$.

10.2.2 Réduction du problème d'ordonnancement au problème d'empaquetage restreint

Le principe est d'autoriser une petite erreur sur le calcul du temps d'exécution minimum pour pouvoir ramener le problème à la restriction de l'empaquetage que l'on sait résoudre en temps polynomial. Il y aura deux sources d'erreur :

- l'arrondi des tailles des objets pour les ramener dans un ensemble fini ;
- l'arrêt de la recherche dichotomique qui garantit un temps de calcul polynomial.

Nous pouvons réduire à volonté ces erreurs, au prix d'une augmentation du temps de calcul. Et pour une précision fixée, le temps de calcul sera polynomial en n. Nous obtiendrons donc bien un schéma d'approximation polynomial.

Soient ε le paramètre de précision et $t \in [\mathrm{LB}, 2\ \mathrm{LB}]$. Nous dirons qu'un objet est *petit* si sa taille est inférieure à $t\varepsilon$; nous omettons ces objets pour l'instant. Les tailles des autres objets sont arrondies vers le bas comme suit : pour chaque j, soit $i \geq 0$ tel que $p_j \in \left[t\varepsilon(1+\varepsilon)^i,\ t\varepsilon(1+\varepsilon)^{i+1}\right[$, remplacer p_j par $p'_j = t\varepsilon(1+\varepsilon)^i$. Les p'_j prennent au plus $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$ valeurs distinctes. Calculons un empaquetage optimal pour les objets arrondis dans des boîtes de taille t avec notre algorithme à base de programmation dynamique. Puisque l'arrondi réduit la taille des objets d'un facteur au plus $1+\varepsilon$, l'empaquetage calculé est valide pour les objets avec leurs tailles réelles dans des boîtes de taille $t\cdot(1+\varepsilon)$. Partons de cet empaquetage des objets réels dans des boîtes de taille $t\cdot(1+\varepsilon)$ et plaçons-y les petits objets de façon gloutonne, en n'ouvrant de nouvelles boîtes que si c'est nécessaire. Remarquons que si une nouvelle boîte est nécessaire, toutes les autres boîtes sont remplies au moins jusqu'à t. Notons $\alpha(I, t, \varepsilon)$ le nombre de boîtes utilisées ; rappelons que ces boîtes ont pour taille $t \cdot (1+\varepsilon)$.

L'algorithme ci-dessus est la *procédure centrale* au cœur du PTAS. Le lemme 10.5 et son corollaire démontrent que cette procédure produit un minorant du temps d'exécution optimal.

Lemme 10.5 $\quad \alpha(I, t, \varepsilon) \leqslant \mathrm{boîtes}(I, t)$.

Preuve : Si l'algorithme n'ouvre pas de nouvelles boîtes lors de l'insertion des petits objets, le résultat est immédiat, car les objets arrondis vers le bas ont été placés de façon optimale dans les boîtes de taille t. Dans le cas contraire, toutes les boîtes sauf la dernière sont remplies jusqu'à t. Ainsi, tout empaquetage de I doit utiliser au moins $\alpha(I, t, \varepsilon)$ boîtes de taille t. □

Mais $\mathrm{OPT} = \min\{t :\ \mathrm{boîtes}(I, t) \leqslant m\}$, le lemme 10.5 implique donc :

Corollaire 10.6 $\quad \min\{t :\ \alpha(I, t, \varepsilon) \leqslant m\} \leqslant \mathrm{OPT}$.

Si une recherche dichotomique pouvait calculer $\min\{t : \alpha(I, t, \varepsilon) \leqslant m\}$ exactement, nous pourrions calculer directement un ordonnancement de temps d'exécution $\leq (1+\varepsilon)\,\mathrm{OPT}$ avec la procédure centrale. Spécifions maintenant cette recherche dichotomique pour en garantir les performances. La recherche dichotomique s'étend sur l'intervalle $[\mathrm{LB}, 2\ \mathrm{LB}]$ de longueur LB. À chaque itération la longueur de l'intervalle est divisée par 2. Nous lançons la recherche jusqu'à ce que l'intervalle soit de longueur $\leq \varepsilon$ LB, ce qui nécessite $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ itérations. Notons T l'extrémité droite de l'intervalle résultant.

Lemme 10.7 $\quad T \leqslant (1+\varepsilon)\ \mathrm{OPT}$.

Preuve : Clairement, $\min\{t :\ \alpha(I, t, \varepsilon) \leqslant m\} \in [T - \varepsilon\ \mathrm{LB}, T]$. Ainsi,

$$T \leq \min\{t : \ \alpha(I,t,\varepsilon) \leqslant m\} + \varepsilon \ \mathrm{LB}\,.$$

Comme $\mathrm{LB} \leqslant \mathrm{OPT}$, le corollaire 10.6 termine la preuve. □

L'empaquetage produit par la procédure principale sur l'entrée $t = T$, donne donc un ordonnancement de temps d'exécution inférieur à $T \cdot (1+\varepsilon)$. Nous en concluons :

Théorème 10.8 *L'algorithme décrit ci-dessus produit un ordonnancement valide de temps d'exécution inférieur à*

$$(1+\varepsilon)^2 \ \mathrm{OPT} \leqslant (1+3\varepsilon) \ \mathrm{OPT}\,.$$

Le temps de calcul de l'algorithme est $O\left(n^{2k}\lceil \log_2 \frac{1}{\varepsilon}\rceil\right)$, où $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon}\rceil$.

10.3 Exercices

10.1 (Graham [121]) L'instance critique proposée dans l'exemple 10.4 pour la 2-approximation utilise une tâche très longue. Cela suggère l'amélioration suivante : commencer par trier les tâches par durée décroissante avant de les ordonnancer. Démontrez que ce nouvel algorithme est une 4/3-approximation. Exhibez une instance critique.

10.2 (Horowitz et Sahni [139]) Proposez un FPTAS pour la variante du problème de l'ordonnancement de temps d'exécution total minimum, où le nombre de machines m est une constante fixée.

10.4 Notes

L'algorithme 10.2 fut proposé par Graham [120]. Le PTAS est dû à Hochbaum et Shmoys [136].

11 Voyageur de commerce euclidien

Nous présentons dans ce chapitre, un PTAS pour le voyageur de commerce dans le cas particulier où les points appartiennent à un espace euclidien de dimension d. Comme précédemment, l'idée principale est de « lisser » l'instance en fonction de la précision ε souhaitée puis de résoudre l'instance lissée par programmation dynamique. Une particularité de cet algorithme est que l'on ne sait pas construire *directement* (c'est-à-dire à la simple lecture de l'instance) une bonne instance rabotée pour ce problème de façon déterministe – sa construction directe est probabiliste (mais peut être dérandomisée par énumération exhaustive).

Problème 11.1 (TSP euclidien) Pour une dimension d fixée, trouver un tour de longueur minimum passant par n points donnés de $\mathbf{R}^d$. La distance entre deux points x et y est la distance euclidienne dans $\mathbf{R}^d$, c'est-à-dire $\left(\sum_{i=1}^{d} (x_i - y_i)^2\right)^{1/2}$.

11.1 L'algorithme

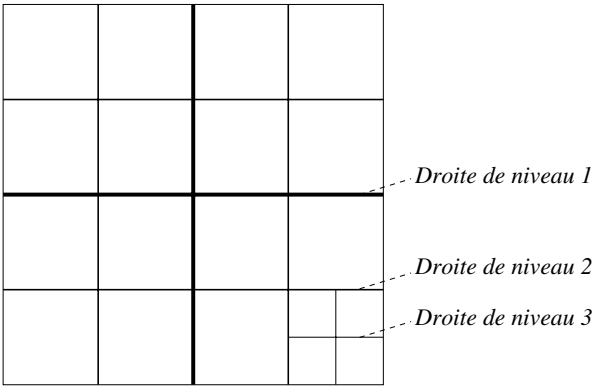
Nous présentons l'algorithme pour le plan, c'est-à-dire $d = 2$. La généralisation à une dimension d arbitraire est immédiate. L'algorithme présente de nombreuses subtilités. Afin de faire ressortir les idées principales, quelques unes d'entre elles seront laissées en exercices.

Nous appellerons *enveloppe carrée* des n points d'une instance, le plus petit carré (le plus en bas à gauche en cas d'*ex æquo*, par exemple) qui contient les n points et dont les côtés sont parallèles aux axes. Quitte à perturber légèrement l'instance, nous pouvons supposer sans perte de généralité que le côté L du carré vaut $4n^2$, et que les points se trouvent aux croisements de la grille unitaire basée sur le carré, c'est-à-dire que les points ont des coordonnées entières dans le repère défini par le carré (voir exercice 11.1). Sans perte de généralité, nous supposerons également que n est une puissance de 2. Posons donc $L = 2^k$ avec $k = 2 + 2\log_2 n$.

La *décomposition élémentaire* de l'enveloppe carrée est définie comme une partition récursive en carrés de plus en plus petits. Le carré de taille $L \times L$ est divisé en quatre carrés de taille $L/2 \times L/2$, ainsi de suite. Cette décomposition se représente commodément par un arbre quaternaire T, dont la racine est

l'enveloppe carrée des points. Les quatre fils de la racine sont les quatre carrés $L/2 \times L/2$, etc. Nous associons à chaque nœud de T son *niveau*. La racine est au niveau 0, ses enfants sont au niveau 1, etc. Nous parlerons abusivement du niveau du carré associé à un nœud. Les carrés de niveau i ont pour taille $L/2^i \times L/2^i$. La décomposition s'arrête lorsqu'on obtient des carrés unitaires. La profondeur de T est clairement $k = O(\log n)$. Tout carré associé à un nœud de T sera qualifié d'*utile*.

Définissons à présent le *niveau des droites horizontales et verticales* intervenant dans la décomposition élémentaire (c'est-à-dire les droites de la grille unitaire basée sur l'enveloppe carrée). Le niveau des deux droites divisant l'enveloppe carrée en quatre carrés est 1. Plus généralement, le niveau des 2^i droites divisant les carrés de niveau $i-1$ en carrés de niveau i, est i. Ainsi, les arêtes contenues dans chaque droite de niveau i appartiennent aux carrés utiles de niveaux $i, i+1, \ldots$ Le plus grand carré utile appuyé sur ces droites est donc de taille $L/2^i \times L/2^i$:



À chaque droite on associe un ensemble de points spéciaux, appelés *portes*. La solution approchée que nous recherchons n'est autorisée à couper les droites qu'en leurs portes. Les portes d'une droite sont régulièrement espacées. Pour une droite de niveau i, les portes sont placées tous les $L/(2^i m)$, où le paramètre m est une puissance de 2 choisie dans l'intervalle $[k/\varepsilon, 2k/\varepsilon]$. Clairement, $m = O(\log n/\varepsilon)$. Puisque le plus grand carré appuyé sur une droite de niveau i a pour taille $L/2^i \times L/2^i$, tout carré utile dispose d'au plus $4m$ portes sur son bord. Le choix d'une puissance de 2 pour m, assure que toutes les portes d'un carré de niveau donné sont des portes des carrés de niveaux supérieurs qui les contiennent.

Nous dirons qu'un tour τ est *conforme* à la décomposition élémentaire si c'est un tour des n points et d'un sous-ensemble quelconque de portes. Un tel tour est autorisé à passer plusieurs fois par une même porte, mais n'est autorisé à se couper lui-même en aucun autre point. La propriété structurelle clé est qu'il existe un tour conforme de longueur inférieure à $(1+\varepsilon) \cdot \mathrm{OPT}$. L'argument est probabiliste, nous le verrons plus tard. Voyons tout d'abord comment le PTAS en découle.

Lemme 11.2 *Pour tout tour τ conforme à la décomposition élémentaire, il existe un tour τ' conforme à la décomposition, de longueur inférieure ou égale à celle de τ, et qui ne passe que deux fois, au plus, par une même porte.*

Preuve : L'idée est que l'élimination des auto-intersections en « court-circuitant » le tour ne peut qu'améliorer la longueur de ce dernier, grâce à l'inégalité triangulaire (vérifiée par la distance euclidienne). Si τ utilise une porte de la droite l plus de deux fois, nous pouvons court-circuiter τ des deux côtés de l jusqu'à ce qu'il n'emprunte la porte que deux fois. Une telle opération peut engendrer des intersections supplémentaires, qui peuvent à leur tour être supprimées de la même façon. □

Lemme 11.3 *Il existe un algorithme qui calcule un tour conforme optimal pour la décomposition élémentaire en temps $2^{O(m)} = n^{O(1/\varepsilon)}$.*

Preuve : Nous allons utiliser la programmation dynamique pour remplir une table qui stocke le coût optimal de toutes les visites valides de chaque carré utile dans un tour conforme. Nous esquisserons juste la preuve dont les détails sont présentés dans l'exercice 11.2.

Notons τ le tour optimal conforme désiré. D'après le lemme 11.2, nous pouvons supposer que τ passe au plus deux fois par chaque porte. Le nombre total d'entrées et de sorties de τ dans un carré utile S est donc majoré par $8m$. La partie de τ interne à S se compose d'au plus $4m$ chemins, reliant chacun une porte d'entrée à une porte de sortie de S, et couvrant ensemble tous les points à l'intérieur de S. De plus, ces chemins ne peuvent se couper qu'en leurs extrémités. Cela signifie que si l'on aligne les portes du carré sur une droite et qu'on place sur chaque porte d'entrée d'un segment une parenthèse ouvrante, et sur chaque porte de sortie d'un segment une parenthèse fermante, nous obtenons une expression correctement parenthésée.

Visite non-valide *Visite valide*

Un tel appariement orienté des portes sera appelé une *visite valide* de S.

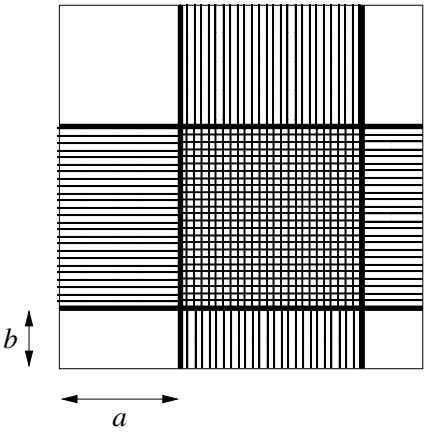
Le nombre de carrés utiles est polynomial en n. Démontrons que le nombre de visites valides d'un carré utile est majoré par $n^{O(1/\varepsilon)}$; le nombre d'entrées de la table sera donc bien polynomial, majoré par $n^{O(1/\varepsilon)}$.

Soit S un carré utile. Chacune de ses portes est utilisée 0, 1, ou 2 fois, soit un total de $3^{4m} = n^{O(1/\varepsilon)}$ possibilités. Ne retenons parmi celles-ci que celles qui utilisent un nombre pair de portes. Une telle configuration utilise $2r$ portes. Calculons à présent le nombre d'appariements valides possibles. Il s'agit du nombre d'expressions correctement parenthésées de longueur $2r$. Ce nombre est donné par le r-ième nombre de Catalan, qui est majoré par $2^{2r} = n^{O(1/\varepsilon)}$. Par conséquent, le nombre total de visites valides de S est majoré par $n^{O(1/\varepsilon)}$.

Pour chaque entrée de la table, nous devons calculer la longueur optimale de la visite valide correspondante. La table est construite en remontant l'arbre de décomposition, à partir des feuilles. Soit V une visite valide d'un carré S de niveau i. V fixe les portes d'entrée et de sortie aux bords de S. Le carré de S a quatre enfants de niveau $i+1$, dont quatre des côtés sont à l'intérieur de S et qui introduisent au plus $4m$ portes supplémentaires au total. Chacune de ces portes est utilisée 0, 1 ou 2 fois, offrant ainsi un total de $n^{O(1/\varepsilon)}$ possibilités. Prenons une de ces possibilités, et étudions toutes les utilisations possibles de ces portes par une visite valide V. Il s'agit de calculer tous les appariements valides des portes, consistants avec la visite V ; leur nombre est de nouveau majoré par le nombre de Catalan, donc par $n^{O(1/\varepsilon)}$. Chacun de ces appariements détermine une visite valide dans chacun des quatre carrés. Les coûts optimaux de ces quatres visites valides ayant été déjà calculés, il suffit de les sommer. La plus petite de ces sommes détermine la façon optimale d'effectuer la visite valide V du carré S. □

11.2 Correction de l'algorithme

Pour démontrer que l'algorithme est correct, il suffit de montrer qu'il existe un tour conforme à la décomposition élémentaire et de longueur inférieure à $(1+\varepsilon)\,\mathrm{OPT}$. En fait, ce n'est pas toujours le cas (voir exercice 11.3). Nous allons cependant exhiber un ensemble de décompositions élémentaires, tel qu'au moins la moitié de ces décompositions admette un tour conforme de la bonne longueur, indépendamment des positions des n points. La sélection d'une décomposition aléatoire dans cet ensemble terminera donc la preuve.



Nous allons définir L^2 décompositions distinctes de l'enveloppe carrée, obtenues par translations unitaires de la décomposition élémentaire. Pour chaque entier a et b, où $0 \leqslant a, b < L$, nous appellerons décomposition (a, b), la

décomposition obtenue en translatant chaque droite verticale de sa position initiale x à la position $(a + x) \bmod L$, et en translatant chaque droite horizontale de sa position initiale y à la position $(b + y) \bmod L$. Par exemple, les droites centrales sont déplacées aux coordonnées $(a + L/2) \bmod L$ et $(b + L/2) \bmod L$, respectivement.

L'enveloppe carrée est vue comme un tore $L \times L$, et tout carré utile qui s'étend maintenant au-delà des bords doit être pensé comme un unique carré sur ce tore. Notons que les coordonnées des n points restent inchangées et que seules les décompositions sont translatées.

Notons π un tour optimal et $N(\pi)$ le nombre total de fois que π croise une droite horizontale ou verticale de la grille. Si π passe par un point à l'intersection de deux droites de la grille, ce point est compté deux fois. La démonstration du fait suivant est laissée à l'exercice 11.4.

Lemme 11.4 $N(\pi) \leqslant 2 \cdot \mathrm{OPT}$.

Le théorème ci-dessous est la pierre angulaire du PTAS.

Théorème 11.5 *Pour a et b tirés au hasard uniformément dans $[0, L[$, l'espérance du surcoût induit pour rendre π conforme à la décomposition (a, b) est inférieure à $2\varepsilon \cdot \mathrm{OPT}$.*

Preuve : Considérons une décomposition quelconque et étudions comment conformer π à cette décomposition. Il s'agit de remplacer tous les segments de π qui traversent une droite l de la grille, par deux segments qui traversent l par la porte la plus proche du point d'intersection. La variation induite de la longueur du tour est majorée par la distance entre deux portes consécutives de l.

Étudions maintenant l'espérance de la variation de la longueur du tour induite par l'élimination d'un des croisements de π avec une droite de la grille. Notons l cette droite. Le niveau de l est i avec probabilité $2^i/L$ dans la décomposition aléatoire ; dans ce cas, la distance entre deux portes consécutives de l vaut $L/(2^i m)$. Par conséquent, l'espérance de la variation induite de la longueur du tour est majorée par

$$\sum_i \frac{L}{2^i m} \cdot \frac{2^i}{L} = \frac{k}{m} \leqslant \varepsilon,$$

car $m \in [k/\varepsilon, 2k/\varepsilon]$. En sommant sur tous les $N(\pi)$ croisements potentiels, puis en appliquant le lemme 11.4, nous en concluons le théorème. □

Remarque 11.6 On peut résumer les idées conduisant au théorème 11.5 ainsi. Les droites de bas niveau sont adjacentes aux plus grands carrés utiles. Leurs portes doivent donc être très espacées, afin d'assurer que tout carré utile n'ait pas plus de $4m$ portes (fait nécessaire pour que la programmation

dynamique termine en temps polynomial). Cependant, il devient alors possible d'exhiber des instances pour lesquelles il n'existe pas de tour conforme à la décomposition élémentaire, qui soit de la bonne longueur (exercice 11.3). Mais, comme il y a relativement peu de droites de bas niveau, ce phénomène est peu probable, ce qu'exploite le théorème 11.5.

Enfin, l'inégalité de Markov donne :

Corollaire 11.7 *Pour a et b tirés au hasard uniformément dans $[0, L[$, la probabilité qu'il existe un tour conforme à la décomposition (a, b) de longueur inférieure à $(1 + 4\varepsilon) \cdot \mathrm{OPT}$, est supérieure à $1/2$.*

Remarquons que les lemmes 11.2 et 11.3 s'appliquent également aux décompositions translatées (a, b). Voici donc le PTAS : choisir uniformément une décomposition aléatoire, puis trouver un tour optimal conforme à cette décomposition par programmation dynamique (lemme 11.3). Notons qu'il est possible de dérandomiser cet algorithme, en essayant toutes les translations possibles (il y en a un nombre polynomial) et en ne conservant que le tour le plus court. Nous en déduisons donc :

Théorème 11.8 *Il existe un PTAS pour le TSP euclidien dans $\mathbf{R}^2$.*

11.3 Exercices

11.1 Démontrez qu'il est possible de supposer, sans perte de généralité, que l'enveloppe carrée a pour côté $L = 4n^2$ et que les points sont placés aux croisements de la grille unitaire basée sur ce carré.
Indication : Comme l'enveloppe carrée est le plus petit carré parallèle aux axes contenant les points, la longueur de son côté est un minorant de OPT. L'allongement du tour optimal lorsqu'on replace chaque point sur la grille est donc inférieur à OPT/n^2.

11.2 Complétez la preuve du lemme 11.3.

11.3 Exhibez une instance du TSP euclidien où le tour optimal s'allonge d'un facteur constant fixé si on le conforme à la décomposition élémentaire.
Indication : Proposez un tour qui croise à de nombreuses reprises la droite centrale de la décomposition (où les portes sont les plus espacées).

11.4 Démontrez le lemme 11.4.
Indication : Remarquez que le membre de gauche de l'inégalité mesure la longueur du tour π pour la distance ℓ_1. La borne $2\sqrt{2} \cdot \mathrm{OPT}$ est plus facile à établir, puisqu'elle est vraie pour chaque arête. Cette borne suffit d'ailleurs pour prouver le PTAS.

11.5 Généralisez le PTAS au voyageur de commerce euclidien dans $\mathbf{R}^d$.

11.6 Généralisez l'algorithme aux autres normes de $\mathbf{R}^d$.

11.7 (Arora [12]) Exhibez un PTAS pour le problème de l'arbre de Steiner euclidien : étant donné n points de $\mathbf{R}^d$, trouver l'arbre de longueur totale minimum passant par les n points (et n'importe quel autre sous-ensemble de points, dits de Steiner). La distance est la distance euclidienne.

11.8 Étudions le problème de l'arbre de Steiner dans $\mathbf{R}^2$. Démontrez que dans tout arbre de Steiner optimal, les arêtes incidentes aux points de Steiner de degré 3, forment des angles de 120° (examinez les dessins de Gauss reproduits en couverture pour une illustration de ce fait).

11.4 Notes

Le premier PTAS pour le TSP euclidien a été proposé par Arora [11], à la suite d'un article de Grigni, Koutsoupias et Papadimitriou [122] donnant un PTAS pour le TSP sur les graphes planaires. Mitchell [216] a obtenu indépendamment le même résultat. Arora [12] a proposé ensuite un algorithme qui produit une solution pour toute dimension d fixée, en temps $n(\log n)^{O(1/\varepsilon)}$. Un PTAS avec un meilleur temps de calcul a enfin été proposé par Rao et Smith [238]. Ce chapitre s'inspire des travaux de Arora [12] et de Arora, Raghavan et Rao [15].

Deuxième partie

Programmation linéaire en algorithmique

12 Introduction à la dualité en programmation linéaire

Une grande partie des algorithmes d'approximation est, actuellement, construite autour de la programmation linéaire.[1] La section 12.1 présente les concepts fondamentaux de cette théorie. La section 12.2 montre comment utiliser le théorème de dualité de la programmation linéaire pour obtenir des relations min-max, dont la portée est très importante en algorithmique. Enfin, la section 12.3 introduit deux techniques algorithmiques fondamentales ainsi qu'une méthode d'analyse d'algorithme : l'arrondi, le schéma primal-dual et la méthode de l'alignement dual. Ces techniques sont à la base de tous les algorithmes de la deuxième partie de ce livre.

12.1 Le théorème de dualité en programmation linéaire

La programmation linéaire consiste à optimiser (maximiser ou minimiser) une fonction linéaire dont les variables sont contraintes par des inégalités linéaires. La fonction à optimiser s'appelle la *fonction objectif*.[2] Un fait particulièrement intéressant est qu'un tel problème est bien caractérisé (voir définitions section 1.2). Illustrons cela sur l'exemple suivant :

$$
\begin{array}{ll}
\text{minimiser} & 7x_1 + x_2 + 5x_3 \\
\text{sous les contraintes} & x_1 - x_2 + 3x_3 \geq 10 \\
& 5x_1 + 2x_2 - x_3 \geq 6 \\
& x_1, x_2, x_3 \geq 0
\end{array}
$$

Remarquons que dans cet exemple, toutes les contraintes sont du type « $\geqslant$ » et que toutes les variables sont contraintes à être positives. Cette forme est la *forme standard* d'un programme linéaire de minimisation ; il existe des transformations élémentaires qui permettent de réécrire tout programme linéaire de minimisation sous cette forme. Les motivations concernant le choix de cette forme apparaîtront clairement par la suite.

[1] *Linear programming (LP)*, en anglais.
[2] *Objective function*, en anglais.

On appelle *solution* d'un programme linéaire, toute affectation des variables (qu'elle vérifie les contraintes ou non). Toute solution du programme linéaire qui satisfait toutes les contraintes, est dite *réalisable*.[3] Notons $\boldsymbol{z}^*$ la valeur optimale de ce programme linéaire. Posons-nous la question « $\boldsymbol{z}^*$ est-il inférieur à α ? » pour un nombre rationnel donné α. Par exemple, demandons-nous si $\boldsymbol{z}^* \leqslant 30$. Un certificat positif pour cette question consiste tout simplement en une solution réalisable dont la valeur objectif est inférieure à 30. Par exemple, $\boldsymbol{x} = (2, 1, 3)$ est un certificat positif pour notre problème : il satisfait les deux contraintes et sa valeur objectif est $7 \cdot 2 + 1 + 5 \cdot 3 = 30$. Remarquons que tout certificat positif pour cette question produit un majorant de $\boldsymbol{z}^*$.

Comment obtenir un certificat négatif pour cette question ? C'est-à-dire, comment générer un bon minorant de $\boldsymbol{z}^*$? Nous pouvons procéder ainsi : puisque les (x_i) sont contraints à être positifs, nous savons par comparaison des coefficients terme à terme que $7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3$. Or, le membre de droite est contraint à être supérieur à 10, la fonction objectif est donc minorée par 10 pour toute solution réalisable. Nous obtenons un meilleur minorant en combinant deux contraintes : pour toute solution réalisable $\boldsymbol{x}$,

$$7x_1 + x_2 + 5x_3 \geq (x_1 - x_2 + 3x_3) + (5x_1 + 2x_2 - x_3) \geq 16.$$

Le principe est donc d'obtenir un minorant en recherchant une combinaison linéaire positive des contraintes telle que le coefficient résultant pour chaque variable x_i soit inférieur (ou égal) aux coefficients correspondants dans la fonction objectif. La constante du membre de droite de l'inégalité obtenue, est alors un minorant de $\boldsymbol{z}^*$ (puisque tous les x_i sont positifs dans toute solution réalisable). Notons qu'il est essentiel que la combinaison linéaire soit positive, afin de ne pas inverser le sens des inégalités.

Pour obtenir le meilleur minorant possible par cette méthode, il s'agit donc de trouver une combinaison linéaire positive des contraintes qui maximise la constante du membre de droite. Ce problème se formule également sous forme d'un programme linéaire !

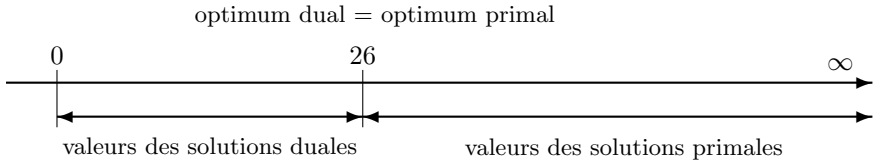
$$\begin{array}{lll}
\text{maximiser} & 10y_1 + 6y_2 & \\
\text{sous les contraintes} & y_1 + 5y_2 & \leq 7 \\
& -y_1 + 2y_2 & \leq 1 \\
& 3y_1 - y_2 & \leq 5 \\
& y_1, y_2 \geq 0 &
\end{array}$$

Les variables y_1 et y_2 sont les coefficients pondérateurs respectifs des première et seconde contraintes de notre exemple. Le programme linéaire initial est appelé le *programme primal*, et le second, *programme dual*.[4]

[3] *Feasible solution*, en anglais.

[4] *Primal program* et *dual program*, en anglais.

L'obtention du programme dual d'un programme linéaire s'automatise très simplement ; et si l'un est un problème de minimisation, l'autre est un problème de maximisation. De plus, le dual du dual est le programme primal (voir exercice 12.1). Par construction, toute solution réalisable du programme dual donne une minoration de la valeur optimale du primal. Remarquons que l'inverse est également vrai : toute solution réalisable du programme primal donne un majorant de la valeur optimale du dual. Par conséquent, si nous trouvons deux solutions réalisables, une du dual et une du primal, ayant la même valeur objectif, ces deux solutions sont optimales. Dans notre exemple, $\boldsymbol{x} = (7/4, 0, 11/4)$ et $\boldsymbol{y} = (2, 1)$ sont des solutions réalisables de valeurs objectifs identiques, 26 ; ce sont donc des solutions optimales (voir figure ci-dessous). Le lecteur pourrait se dire que cet exemple a été construit pour que cela se produise. Or, ce n'est pas une exception, mais bien la règle ! Il s'agit du théorème principal de la programmation linéaire : le *théorème de dualité*.[5]



Pour énoncer le théorème, considérons le problème de minimisation (primal) suivant, mis sous forme standard ; nous pourrions tout aussi bien énoncer le théorème pour un programme primal de maximisation.

$$\begin{aligned} &\text{minimiser} && \sum_{j=1}^{n} c_j x_j && && (12.1)\\ &\text{sous les contraintes} && \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && i = 1, \ldots, m\\ & && x_j \geq 0, && j = 1, \ldots, n \end{aligned}$$

où les a_{ij}, b_i, et c_j sont des rationnels donnés.
Pour tout i, nous associons une variable y_i à la i-ième inégalité. Nous obtenons le programme dual :

$$\begin{aligned} &\text{maximiser} && \sum_{i=1}^{m} b_i y_i && && (12.2)\\ &\text{sous les contraintes} && \sum_{i=1}^{m} a_{ij} y_i \leq c_j, && j = 1, \ldots, n\\ & && y_i \geq 0, && i = 1, \ldots, m \end{aligned}$$

[5] *LP-duality theorem*, en anglais.

Théorème 12.1 (Dualité en programmation linéaire) *Un programme linéaire (primal) possède une solution optimale finie ssi son programme dual en possède une également. De plus, si* $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$ *et* $\boldsymbol{y}^* = (y_1^*, \ldots, y_m^*)$ *sont deux solutions optimales des programmes primal et dual respectivement, alors*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Remarquons que le théorème de dualité est aussi une relation min-max, puisqu'il établit une égalité entre un problème de minimisation et un problème de maximisation. Un corollaire de ce théorème est que la résolution de la programmation linéaire est un problème bien caractérisé. Les solutions réalisables du primal (resp. dual) produisent des certificats positifs (resp. négatifs) en réponse à la question « La valeur optimale est-elle inférieure ou égale à α ? ». Une conséquence de ce théorème est que la résolution d'un programme linéaire est dans $\mathbf{NP} \cap \text{co-}\mathbf{NP}$.

Revenons à notre exemple. Par construction, toute solution réalisable du dual donne un minorant de la valeur optimale du primal. C'est le sens facile du théorème de dualité, appelé parfois le *théorème de dualité faible*.[6] Nous allons donner une preuve formelle de cette inégalité, qui sera fondamentale pour la suite. Plusieurs algorithmes exacts reposent sur le théorème de dualité de la programmation linéaire (fort). Cependant, en algorithmique d'approximation, la forme faible suffit la plupart du temps.

Théorème 12.2 (Théorème de dualité faible en programmation linéaire) *Si on considère des solutions réalisables* $\boldsymbol{x} = (x_1, \ldots, x_n)$ *du programme primal, et* $\boldsymbol{y} = (y_1, \ldots, y_m)$ *du dual, alors*

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i. \tag{12.3}$$

Preuve : Puisque $\boldsymbol{y}$ est une solution réalisable du dual et que les (x_j) sont positifs,

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j. \tag{12.4}$$

De même, puisque $\boldsymbol{x}$ est une solution réalisable du primal et que les (y_i) sont positifs,

[6] *Weak duality theorem*, en anglais.

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}x_j\right) y_i \geq \sum_{i=1}^{m} b_i y_i. \tag{12.5}$$

Nous terminons la preuve en remarquant que

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_{ij}y_i\right) x_j = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}x_j\right) y_i.$$

□

D'après le théorème de dualité, $\boldsymbol{x}$ et $\boldsymbol{y}$ sont toutes deux des solutions optimales ssi (12.3) est une égalité, c'est-à-dire ssi (12.4) et (12.5) sont des égalités. Nous en concluons le résultat structurel suivant sur l'espace des solutions optimales :

Théorème 12.3 (Théorème des écarts complémentaires)[7] *Étant données $\boldsymbol{x}$ et $\boldsymbol{y}$ deux solutions réalisables du primal et du dual respectivement, alors $\boldsymbol{x}$ et $\boldsymbol{y}$ sont toutes deux optimales ssi les conditions suivantes sont toutes satisfaites :*

Conditions primales des écarts complémentaires
Pour tout $1 \leq j \leq n$: $x_j = 0$ ou $\sum_{i=1}^{m} a_{ij}y_i = c_j$; et

Conditions duales des écarts complémentaires
Pour tout $1 \leq i \leq m$: $y_i = 0$ ou $\sum_{j=1}^{n} a_{ij}x_j = b_i$.

Le théorème des écarts complémentaires joue un rôle primordial dans la conception d'algorithmes efficaces de résolution de nombreux problèmes ou d'approximation de leurs solutions ; se référer au chapitre 15 pour plus de détails. Nous recommandons tout particulièrement au lecteur d'étudier les algorithmes de résolution du problème du couplage pondéré mentionnés dans les notes section 12.5.

12.2 Relations min-max et dualité en programmation linéaire

Afin de mesurer l'importance de la théorie de la dualité de la programmation linéaire en algorithmique d'approximation, il est utile de bien comprendre son rôle en algorithmique exacte. Nous allons donc revoir certains de ces principes en prenant l'exemple du théorème du flot maximum et de la coupe minimum.[8] Nous démontrerons en particulier, comment cette relation min-max et d'autres encore dérivent du théorème de dualité en programmation linéaire. Certaines des idées développées ici pour les coupes et les flots

[7] *Complementary slackness conditions*, en anglais.
[8] *Max-flow min-cut theorem*, en anglais.

seront utilisées par la suite pour l'étude des multiflots[9] aux chapitres 18, 20, et 21.

Le problème du flot maximum dans un réseau est le suivant : étant donné un graphe orienté[10] $G = (V, E)$ ayant deux sommets distingués, la *source* s et le *puits* t, et muni de capacités positives $c : E \to \mathbf{R}^+$ sur ses arcs, trouver le flot maximum qui puisse être envoyé de s à t, tout en vérifiant :

1. *Contraintes de capacité :* Pour tout arc e, le flot qui traverse e est majoré par la capacité de e, et
2. *Conservation du flot :* Pour tout sommet v, autre que s et t, le flot total entrant dans v doit être égal au flot sortant de v.

Une *coupe de s à t*[11] est définie par une partition des sommets en deux ensembles X et $\overline{X}$ telle que $s \in X$ et $t \in \overline{X}$; elle consiste en l'ensemble des arcs allant de X à $\overline{X}$. La capacité $c(X, \overline{X})$ de la coupe $(X, \overline{X})$ est la somme des capacités de ses arcs. Du fait des contraintes de capacité, la capacité de toute coupe de s à t majore la valeur de tout flot valide. Par conséquent, si la capacité d'une coupe de s à t est égale à la valeur d'un flot valide, alors cette coupe est une coupe minimum de s à t et le flot est maximum pour G. Le théorème du flot maximum et de la coupe minimum affirme qu'il existe toujours un flot et une coupe de s à t ayant la même valeur.

Réécrivons le problème du flot maximum sous forme d'un programme linéaire. Commençons par introduire un arc fictif de capacité infinie de t à s, pour transformer le problème de flot de s à t en un problème de circulation de flot maximum ; l'objectif est maintenant de maximiser le flot f_{ts} qui traverse cet arc. Cette modification permet de symétriser le problème en soumettant s et t aussi aux contraintes de flot. Notons f_{ij} le flot traversant l'arc $(i, j) \in E$ et reformulons le problème du flot maximum ainsi :

$$
\begin{array}{lll}
\text{maximiser} & f_{ts} & \\
\text{sous les contraintes} & f_{ij} \leq c_{ij}, & (i,j) \in E \\
& \displaystyle\sum_{j:\ (j,i)\in E} f_{ji} \ - \sum_{j:\ (i,j)\in E} f_{ij} \leq 0, & i \in V \\
& f_{ij} \geq 0, & (i,j) \in E
\end{array}
$$

Le deuxième jeu d'inégalités impose que le flot entrant dans chaque nœud i soit égal au flot sortant de i. En effet si ces inégalités sont satisfaites en chacun des nœuds, alors ce sont toutes des égalités, c'est-à-dire que le flot est bien conservé en chaque nœud (une fuite en l'un des nœuds entraînerait une

[9] *Multicommodity flow*, en anglais.

[10] Le problème du flot maximum pour un graphe non orienté se ramène à ce même problème dans un graphe orienté, en remplaçant chaque arête uv par deux arcs $(u \to v)$ et $(v \to u)$, ayant chacun la même capacité que uv.

[11] *s–t cut*, en anglais.

accumulation sur un autre). Cette astuce permet d'obtenir un programme linéaire sous forme standard de petite taille.

Pour construire le programme dual, nous introduisons les variables d_{ij} et p_i correspondant aux deux jeux de contraintes du primal. Nous interprétons ces variables comme des *indicateurs de distance*[12] sur les arcs et des *potentiels* sur les sommets, respectivement. Voici le programme dual :

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{(i,j)\in E} c_{ij} d_{ij} & \\ \text{sous les contraintes} & d_{ij} - p_i + p_j \geq 0, & (i,j) \in E \\ & p_s - p_t \geq 1 & \\ & d_{ij} \geq 0, & (i,j) \in E \\ & p_i \geq 0, & i \in V \end{array} \tag{12.6}$$

Pour bien comprendre le programme dual, commençons par le transformer en programme à variables entières, à valeurs dans $\{0,1\}$:

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{(i,j)\in E} c_{ij} d_{ij} & \\ \text{sous les contraintes} & d_{ij} - p_i + p_j \geq 0, & (i,j) \in E \\ & p_s - p_t \geq 1 & \\ & d_{ij} \in \{0,1\}, & (i,j) \in E \\ & p_i \in \{0,1\}, & i \in V \end{array} \tag{12.7}$$

Notons $(\boldsymbol{d}^*, \boldsymbol{p}^*)$ une solution optimale de ce programme en nombres entiers. La seule possibilité pour satisfaire l'inégalité $p_s^* - p_t^* \geq 1$ avec des variables à valeurs dans $\{0,1\}$ est que $p_s^* = 1$ et $p_t^* = 0$. Cette solution définit naturellement une coupe $(X, \overline{X})$ de s à t, où X est l'ensemble des sommets de potentiel 1, et $\overline{X}$ ceux de potentiel 0. Considérons un arc (i,j) avec $i \in X$ et $j \in \overline{X}$. Puisque $p_i^* = 1$ et $p_j^* = 0$, d'après la première contrainte, $d_{ij}^* \geqslant 1$. Mais $d_{ij} \in \{0,1\}$, donc $d_{ij}^* = 1$. Les indicateurs de distance des autres arcs sont libres de prendre pour valeurs 0 ou 1 sans violer la première contrainte ; cependant, par minimalité de la fonction objectif, ces indicateurs sont tous nuls. La valeur optimale de la fonction objectif est donc la capacité de la coupe $(X, \overline{X})$, et $(X, \overline{X})$ est une coupe minimum de s à t.

Le problème en nombres entiers ci-dessus encode donc le problème de la coupe minimum de s à t ! Que dire du programme dual (12.6) ? Nous pouvons le voir comme la relaxation du programme (12.7) où les variables sont autorisées à prendre des valeurs rationnelles. En effet, une première relaxation des contraintes donne $1 \geqslant d_{ij} \geqslant 0$ pour $(i,j) \in E$ et $1 \geqslant p_i \geqslant 0$

[12] *Distance label*, en anglais.

pour $i \in V$. Puis en éliminant les contraintes majorant les indicateurs de distance qui sont superflues (leur suppression ne permet pas de meilleures solutions), nous obtenons bien le programme dual initial. Nous dirons que ce programme est la *relaxation*[13] du programme linéaire en nombres entiers (12.7).

Étudions une coupe C de s à t. Tout chemin de s à t passe par un arc de C. Cette observation nous permet d'interpréter toute solution réalisable du programme dual (12.6) comme une *coupe fractionnaire de s à t*[14] : la somme des indicateurs de distance le long des arcs de tout chemin de s à t est supérieure à 1. En effet, suivons un chemin de s à t ($s = v_0, v_1, \ldots, v_k = t$). La somme des différences des potentiels d'un bout à l'autre du chemin vaut

$$\sum_{i=0}^{k-1} (p_i - p_{i+1}) = p_s - p_t.$$

D'après la première contrainte, la somme des valeurs des indicateurs de distance sur les arcs est supérieure à $p_s - p_t$, et donc $\geqslant 1$. Nous définissions donc la *capacité* d'une coupe fractionnaire de s à t, comme la valeur de la fonction objectif dual sur cette coupe.

En principe, la meilleure coupe fractionnaire de s à t pourrait avoir une capacité inférieure à la meilleure coupe « entière ». Ce n'est pourtant pas le cas. Étudions le polyèdre constitué des solutions réalisables du programme dual (12.6). Nous appelons *solution extrémale*[15] tout sommet de ce polyèdre, c'est-à-dire toute solution qui ne peut pas s'écrire comme combinaison linéaire convexe de deux solutions réalisables distinctes. La théorie de la programmation linéaire affirme que pour toute fonction objectif (c'est-à-dire quelles que soient les capacités des arcs de G), il existe une solution extrémale optimale (en supposant bien sûr qu'une solution réalisable optimale finie existe). On peut démontrer que toutes les solutions extrémales de notre polyèdre sont à coordonnées dans $\{0, 1\}$ (voir exercice 12.7). Par conséquent, la relaxation du programme dual admet toujours une solution entière optimale, qui est donc solution du programme dual (12.7).

D'après le théorème de dualité, la valeur d'un flot maximum de G est donc égale à la capacité minimale d'une coupe fractionnaire de s à t. Comme c'est également la capacité d'une coupe minimum de s à t, nous en déduisons le théorème du flot maximum et de la coupe minimum.

Le théorème du flot maximum et de la coupe minimum est donc un cas particulier du théorème de dualité ; le fait essentiel ici est que tous les sommets du polyèdre dual ont des coordonnées entières. En fait, la plupart des relations min-max en optimisation combinatoire s'expliquent ainsi.

[13] *LP-relaxation*, en anglais.
[14] *Fractional s–t cut*, en anglais.
[15] *Extreme point solution*, en anglais.

Illustrons une dernière fois la puissance du théorème des écarts complémentaires en obtenant des propriétés supplémentaires des solutions optimales des problèmes de flot maximum et de coupe minimum. Soient $\boldsymbol{f}^*$ une solution optimale du programme primal (c'est-à-dire un flot maximum de s à t), $(\boldsymbol{d}^*, \boldsymbol{p}^*)$ une solution optimale entière du programme dual et $(X, \overline{X})$ la coupe de s à t définie par $(\boldsymbol{d}^*, \boldsymbol{p}^*)$. Étudions un arc (i, j) avec $i \in X$ et $j \in \overline{X}$. Nous avons démontré ci-dessus que $d^*_{ij} = 1$. Puisque $d^*_{ij} \neq 0$, la condition duale des écarts complémentaires donne $f^*_{ij} = c_{ij}$. Étudions maintenant un arc (k, l) tel que $k \in \overline{X}$ et $l \in X$. Puisque $p^*_k - p^*_l = -1$, et $d^*_{kl} \in \{0, 1\}$, la contrainte $d^*_{kl} - p^*_k + p^*_l \geq 0$ est une inégalité stricte. La condition des écarts complémentaires primale donne donc : $f^*_{kl} = 0$. Nous en concluons que les arcs allant de X à $\overline{X}$ sont tous saturés par $\boldsymbol{f}^*$ et qu'aucun flot ne traverse les arcs inverses correspondants. Remarquons que le théorème des écarts complémentaires n'était pas essentiel pour démontrer ces propriétés ; ces dernières découlent également du fait que la valeur du flot traversant la coupe $(X, \overline{X})$ est égale à sa capacité.

12.3 Deux techniques algorithmiques fondamentales

Nous allons donc découvrir maintenant pourquoi la programmation linéaire est un outil fondamental en algorithmique d'approximation. De nombreux problèmes d'optimisation combinatoire se réécrivent sous la forme d'un programme linéaire en nombres entiers. La relaxation de ce programme donne naturellement un minorant du coût de la solution optimale. Comme nous l'avons vu au chapitre 1, c'est une étape clé typique pour l'obtention d'un algorithme d'approximation. À l'instar du problème de la coupe minimum de s à t, les solutions réalisables du programme relâché peuvent être interprétées comme des *solutions fractionnaires* du problème initial. Bien entendu, on ne peut s'attendre à ce que les sommets du polyèdre soient à coordonnées entières pour les problèmes **NP**-difficiles. Aussi, notre travail ne sera pas de chercher une solution optimale du programme relâché, mais plutôt une solution entière proche de l'optimal fractionnaire.

Il existe deux techniques fondamentales pour concevoir un algorithme d'approximation à base de programmes linéaires. La première, la plus évidente, est de résoudre le programme linéaire relâché et de transformer la solution fractionnaire obtenue en une solution entière, en essayant de borner le surcoût induit. La garantie de performances est alors obtenue en comparant le coût de la solution entière produite à celui de la solution fractionnaire. Cette technique s'appelle l'*arrondi en programmation linéaire*, ou simplement *arrondi*.[16]

[16] *LP-rounding*, en anglais.

La seconde technique, plus élaborée, s'appelle *schéma primal-dual*.[17] Il s'agit d'utiliser le programme dual du programme linéaire relâché dans le déroulement de l'algorithme. Convenons que le programme primal soit le programme relâché. Cette méthode construit itérativement une solution entière du programme primal et une solution réalisable du dual. Remarquons que toute solution réalisable du dual donne un minorant de OPT. La garantie de performance est obtenue en comparant les valeurs des deux solutions.

Ces deux techniques ont été utilisées intensivement pour obtenir des algorithmes pour des problèmes fondamentaux. Ces méthodes s'illustrent très simplement sur le problème de la couverture par ensembles. Nous le verrons aux chapitres 14 et 15. Les chapitres suivants présentent des utilisations plus élaborées de ces méthodes pour résoudre différents problèmes.

La théorie de la dualité en programmation linéaire permet également d'analyser des algorithmes combinatoires, via la *méthode de l'alignement dual*.[18] Nous proposons au chapitre 13 une nouvelle analyse de l'algorithme glouton pour la couverture par ensembles, fondée sur cette méthode. Cette méthode a été également appliquée au problème du placement d'installations métrique et sans capacité[19] (voir exercice 24.12). Cette technique semble simple à mettre en œuvre et elle devrait avoir d'autres applications.

12.3.1 Comparaison de ces techniques et notion de saut intégral

Le lecteur est en droit de penser que dans la perspective d'un facteur d'approximation, le schéma primal-dual est moins performant que l'arrondi, puisqu'une solution optimale du primal donne un meilleur minorant qu'une solution réalisable du dual. Cependant il n'en est rien. Pour en donner une explication formelle, nous aurons besoin de la notion cruciale de *saut intégral d'une relaxation* d'un programme linéaire.

Étant donné une relaxation d'un programme linéaire pour un problème de minimisation Π, notons $\mathrm{OPT}_f(I)$ le coût d'une solution fractionnaire optimale d'une instance I, c'est-à-dire la valeur optimale de la fonction objectif dans la relaxation. Le *saut intégral*[20] d'une relaxation d'un programme linéaire est la quantité :

$$\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)},$$

c'est-à-dire la borne supérieure, sur toutes les instances, du quotient de la valeur optimale d'une solution entière sur la valeur optimale d'une solution rationnelle. Dans le cas d'un problème de maximisation, le saut intégral est défini comme la borne inférieure de ce quotient. Comme nous l'avons vu

[17] *Primal-dual schema*, en anglais.
[18] *Dual fitting method*, en anglais.
[19] *Metric uncapacitated facility location problem*, en anglais.
[20] *Integrality gap*, plus rarement *integrality ratio*, en anglais.

section 12.2, la plupart des relations min-max sont issues de relaxations de programmes linéaires, qui ont toujours une solution optimale entière. Dans ce cas, le saut intégral vaut clairement 1. Une telle relaxation est dite *exacte*.

Le meilleur facteur d'approximation qu'on puisse espérer en comparant directement le coût de la solution construite par un algorithme au coût optimal d'une solution fractionnaire (ou à une solution réalisable du dual) est le saut intégral de la relaxation (voir exercice 12.5). Il est intéressant de noter que pour de nombreux problèmes, les deux techniques conduisent à des algorithmes dont le facteur d'approximation est égal au saut intégral de la relaxation.

La principale différence de performance entre ces deux techniques est le temps de calcul des algorithmes obtenus. Les algorithmes à base d'arrondi de solution de programmes linéaires doivent trouver une solution optimale du programme linéaire relâché. Comme la résolution de la programmation linéaire est dans $\mathbf{P}$, cette étape s'effectue en temps polynomial (pour autant que le nombre de contraintes de la relaxation soit polynomial). Même lorsque le nombre de contraintes est exponentiel, cette étape peut se résoudre rapidement, si on dispose d'un *oracle séparateur*,[21] c'est-à-dire d'un algorithme polynomial qui pour tout point de $\mathbf{R}^n$ (n est le nombre de variables dans la relaxation) détermine s'il est réalisable (c'est-à-dire satisfait toutes les contraintes) et si non, exhibe une contrainte non satisfaite (se référer aux notes de la section 12.5 pour des références bibliographiques). Dans les deux cas, les temps de calcul sont élevés, parfois même exorbitants pour le second cas. Remarquons que pour certains problèmes, les algorithmes à base d'arrondi ont besoin des propriétés structurelles particulières des solutions extrémales. De telles solutions se construisent également en temps polynomial.

Le schéma primal-dual, quant à lui, offre une marge de manœuvre suffisante pour exploiter la structure combinatoire particulière de chaque problème individuellement. Il permet ainsi d'obtenir de bons temps de calcul. Ce schéma n'est qu'une ligne directrice pour la conception de l'algorithme, dont les détails sont spécifiques à chaque problème. En fait, dans de nombreux cas, une fois l'algorithme obtenu par schéma primal-dual, il est possible de le reformuler sous forme purement combinatoire en se débarrassant de l'échafaudage de programmation linéaire qui a servi à sa conception.

Voici un autre avantage du schéma primal-dual sur l'arrondi — cette fois-ci non quantifiable objectivement. Un algorithme combinatoire est souvent plus malléable qu'un algorithme nécessitant un solveur de programme linéaire. Une fois qu'un problème est résolu par schéma primal-dual, l'algorithme utilisé peut souvent résoudre ses variantes et généralisations. Les exercices des chapitres 22 et 24 en sont des illustrations. Enfin, d'un point de vue pratique, un algorithme combinatoire est plus utile, car il s'adapte plus

[21] *Separation oracle*, en anglais.

facilement à des applications spécifiques. Les modifications y sont plus faciles pour améliorer ses performances sur des jeux de données particuliers.

12.4 Exercices

12.1 Démontrez que le dual du dual d'un programme linéaire est bien le programme initial.

12.2 Démontrez que tout problème de minimisation peut être réécrit sous une forme standard équivalente, c'est-à-dire sous la forme du programme linéaire (12.1) (avec un accroissement au plus linéaire du nombre de variables et de contraintes).

12.3 Transformez quelques unes des contraintes du programme primal (12.1) en égalités, c'est-à-dire pour un certain ensemble d'indices I,

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, i \in I.$$

Démontrez que dans le programme dual modifié correspondant (12.2), les variables $(y_i)_{i\in I}$ ne sont plus contraintes,[22] c'est-à-dire qu'elles ne sont pas contraintes à être positives. Montrez également que si des variables $(x_j)_{j\in J}$ du programme (12.1) ne sont pas contraintes, alors les contraintes correspondantes sont des égalités dans le dual.

12.4 Considérons les programmes linéaires (13.2) et (13.3), c'est-à-dire la relaxation et le dual du problème de la couverture par ensemble, problème 2.1. Soient $\boldsymbol{x}$ et $\boldsymbol{y}$ deux solutions réalisables primale et duale, respectivement, vérifiant toutes les conditions des écarts complémentaires. Démontrez que la solution duale « paie » exactement le prix de la solution primale, via le mécanisme de « paiement local » suivant : si chaque élément e paie $y_e x_S$ à chaque ensemble S qui contient e, alors la somme d'argent totale collectée par chaque ensemble S est exactement $c(S)x_S$. En déduire que $\boldsymbol{x}$ et $\boldsymbol{y}$ ont la même valeur objectif.

12.5 L'affirmation suivante est-elle exacte ? « Un algorithme d'approximation conçu par relaxation d'un programme linéaire ne peut pas garantir un facteur d'approximation meilleur que le saut intégral de la relaxation. »
Indication : Notons α le saut intégral de cette relaxation. En principe, même si une instance a un mauvais saut intégral, il n'est pas impossible que des propriétés structurelles particulières permettent de montrer que le coût de

[22] *Unconstrained*, en anglais.

la solution trouvée par l'algorithme soit bien inférieur à $\alpha\,\mathrm{OPT}$. Remarquons cependant que si l'instance a un mauvais saut intégral, le coût de la solution trouvée ne peut pas être très inférieur à $\alpha\,\mathrm{OPT}_f$.

12.6 À partir du théorème du flot maximum et de la coupe minimum, démontrez le théorème de Menger :

Théorème 12.4 *Pour tout graphe orienté $G = (V, E)$ ayant deux sommets distingués s et t, le nombre maximum de chemins arcs-disjoints (resp. sommets-disjoints) de s à t est égal au nombre minimum d'arcs (resp. sommets) dont le retrait déconnecte s de t.*

12.7 Démontrez que toute solution extrémale du programme (12.6) a ses coordonnées dans $\{0, 1\}$ et représente donc une coupe valide.
Indication : Une matrice A de taille $n \times m$ est dite *totalement unimodulaire*[23] si les déterminants de toutes ses sous-matrices carrées sont dans $\{1, -1, 0\}$. Démontrez par induction que la matrice de contraintes de ce programme linéaire est totalement unimodulaire. Utilisez le fait qu'une solution réalisable d'un ensemble de contraintes est extrémale ssi n contraintes linéairement indépendantes sont des égalités pour cette solution.

12.8 Nous proposons ici une preuve du théorème de König-Egerváry (théorème 1.6). Soit $G = (V, E)$ un graphe biparti.

1. Démontrez que la relaxation suivante du problème du couplage maximum dans G est exacte (c'est-à-dire admet toujours une solution optimale entière) :

$$\begin{aligned} &\text{maximiser} && \sum_e x_e && && (12.8)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e \leqslant 1, && v \in V\\ & && x_e \geqslant 0, && e \in E \end{aligned}$$

   Indication : Utilisez la même technique que pour l'exercice 12.7. Montrez que toute solution extrémale du programme (12.8) a ses coordonnées dans $\{0, 1\}$, et représente donc un couplage valide.
2. Construisez le programme dual correspondant et démontrez que c'est une relaxation exacte du problème de la couverture par sommets minimum dans le graphe biparti G.
3. Utilisez le résultat précédent pour en déduire le théorème de König-Egerváry.

[23] *Totally unimodular matrix*, en anglais.

12.9 (Edmonds [75])

1. Soit $G = (V, E)$ un graphe non orienté, avec des poids w_e sur les arêtes. La relaxation suivante du problème du couplage de poids maximum dans G est exacte :

$$\begin{aligned} &\text{maximiser} && \sum_e w_e x_e && && (12.9)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e \leqslant 1, && v \in V\\ & && \sum_{e\,:\,e \subset S} x_e \leqslant \frac{|S|-1}{2}, && S \subset V, |S| \text{ impair}\\ & && x_e \geqslant 0, && e \in E \end{aligned}$$

   $e \subset S$ signifie que les deux extrémités de e sont dans S. Construisez le dual de ce programme. On sait que si la fonction de poids est à valeurs entières, le dual est également exact. Démontrez que le théorème 1.7 est une conséquence de ces résultats.

2. Supposons que $|V|$ est pair. La relaxation suivante du problème du couplage parfait de poids minimum dans G est exacte (un couplage est *parfait* s'il couvre tous les sommets). Construisez le dual de ce programme et utilisez le théorème des écarts complémentaires pour donner les conditions satisfaites par toute paire de solutions optimales du primal (entière) et dual dans les deux formulations.

$$\begin{aligned} &\text{minimiser} && \sum_e w_e x_e && && (12.10)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e = 1, && v \in V\\ & && \sum_{e\,:\,e \in S} x_e \leqslant \frac{|S|-1}{2}, && S \subset V, |S| \text{ impair}\\ & && x_e \geqslant 0, && e \in E \end{aligned}$$

12.10 (Edmonds [78]) Démontrez que le programme linéaire suivant encode le problème de l'arbre couvrant de poids minimum.[24] On se donne un graphe $G = (V, E)$, $|V| = n$, muni d'une fonction de coût $c : E \to \mathbf{Q}^+$ sur les arêtes. Pour tout $A \subseteq E$, on note $\kappa(A)$ le nombre de composantes connexes du sous-graphe $G_A = (V, A)$.

[24] *Minimum spanning tree (MST)*, en anglais.

$$\begin{aligned}
&\text{minimiser} && \sum_e c_e x_e && && (12.11)\\
&\text{sous les contraintes} && \sum_{e\in A} x_e = n - \kappa(A), && A \subset E\\
& && \sum_{e\in E} x_e = n-1 && \\
& && x_e \in \{0,1\}, && e \in E
\end{aligned}$$

Dans la suite de l'exercice, nous nous proposons de démontrer que la relaxation de ce programme en nombres entiers est exacte pour le problème du MST.

1. Commençons par réécrire la fonction objectif du programme en nombres entiers (12.11) en la remplaçant par $\max \sum_e -c_e x_e$. Construisez la relaxation et le dual de cette reformulation.
2. Étudions la solution du primal construite par l'algorithme de Kruskal. Notons $e_1, \ldots, e_m$ les arêtes du graphe, triées par poids croissants ($|E| = m$). Cet algorithme extrait un sous-graphe acyclique de cette liste triée de manière gloutonne. Exhibez une solution réalisable duale qui satisfasse avec la solution gloutonne les conditions des écarts complémentaires.

   Indication : Posez $A_t = \{e_1, \ldots, e_t\}$ et considérez la variable duale $\boldsymbol{y}$ suivante : $y_{A_t} = c(e_{t+1}) - c(e_t)$, pour $1 \leqslant t < m$, et $y_E = -c(e_m)$.
3. Démontrez que $\boldsymbol{x}$ est une solution réalisable du programme primal ssi c'est une solution réalisable du programme suivant. C'est-à-dire, démontrez que ce programme est aussi une relaxation exacte du problème du MST.

$$\begin{aligned}
&\text{minimiser} && \sum_e c_e x_e && && (12.12)\\
&\text{sous les contraintes} && \sum_{e\in S} x_e \leqslant |S| - 1, && S \subset V\\
& && \sum_{e\in E} x_e = n-1 && \\
& && x_e \geqslant 0, && e \in E
\end{aligned}$$

12.11 Nous nous proposons dans cet exercice de déduire du théorème de dualité en programmation linéaire le théorème minimax de von Neumann en théorie des jeux. Un *jeu fini de somme nulle à deux joueurs*[25] est défini par une matrice A de taille $m \times n$ à coefficients réels. À chaque tour, le premier joueur, L, choisit une ligne, disons la i-ième ; et simultanément l'autre joueur, C, choisit une colonne, disons la j-ième. Le *gain*[26] du joueur L à la fin du

[25] *A finite two-person zero-sum game*, en anglais.
[26] *Payoff*, en anglais.

tour est le coefficient a_{ij} de la matrice. Ainsi, $|a_{ij}|$ est la somme versée par C à L (resp. par L à C) si a_{ij} est positif (resp. négatif) ; le tour est nul si a_{ij} est nul. L'appellation *jeu de somme nulle* signifie que la somme des fortunes de L et C reste inchangée au cours du temps.

Chaque joueur suit une *stratégie* définie par un vecteur unitaire positif, dont la i-ième coordonnée représente la probabilité qu'il choisisse la ligne ou la colonne i. Notons $\boldsymbol{x}$ et $\boldsymbol{y}$ les vecteurs (de tailles n et m) représentant respectivement les stratégies de L et de C. Suivant ces stratégies, l'espérance du gain de L à chaque tour est $\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$. Chaque joueur recherche la stratégie qui *garantit* une espérance de gain maximum (ou une espérance de perte minimum) et ce, quelle que soit la stratégie adoptée par l'adversaire. Si L opte pour la stratégie $\boldsymbol{x}$, il n'est assuré de gagner qu'au plus $\min_{\boldsymbol{y}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$, où le minimum est calculé sur toutes les stratégies $\boldsymbol{y}$ possibles pour C. Par conséquent, le meilleur choix pour L est un $\boldsymbol{x}$ qui réalise $\max_{\boldsymbol{x}} \min_{\boldsymbol{y}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$. De même, C va minimiser ses pertes en choisissant la stratégie donnée par $\min_{\boldsymbol{y}} \max_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$. Le théorème minimax affirme que pour toute matrice $\boldsymbol{A}$, $\max_{\boldsymbol{x}} \min_{\boldsymbol{y}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y} = \min_{\boldsymbol{y}} \max_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$.

Nous dirons qu'une stratégie est *pure* si elle ne sélectionne qu'une unique ligne ou colonne, c'est-à-dire si le vecteur correspondant n'a qu'un unique 1 et pour le reste, des 0. Une observation clé est que pour toute stratégie $\boldsymbol{x}$ de L, $\min_{\boldsymbol{y}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$ est réalisé par une stratégie pure de C. En effet, soit $\boldsymbol{y}$ une stratégie qui réalise ce minimum. Définissons une stratégie pure en choisissant n'importe laquelle des entrées non nulles de $\boldsymbol{y}$. Comme $\boldsymbol{y}$ est unitaire et positif, il est facile de montrer que cette stratégie pure réalise également le minimum. Ainsi, la stratégie optimale de L est donnée par $\max_{\boldsymbol{x}} \min_j \sum_{i=1}^m a_{ij} x_i$. Le second fait important est que le calcul de la stratégie optimale de L s'exprime sous forme du programme linéaire suivant :

$$
\begin{array}{lll}
\text{maximiser} & z & \\
\text{sous les contraintes} & z - \displaystyle\sum_{i=1}^{m} a_{ij} x_i \leq 0, & j = 1, \ldots, n \\
& \displaystyle\sum_{i=1}^{m} x_i = 1 & \\
& x_i \geq 0, & i = 1, \ldots, m
\end{array}
$$

Construisez le dual de ce programme et démontrez qu'il calcule une stratégie optimale pour C (utilisez le fait que pour toute stratégie $\boldsymbol{y}$ de C, $\max_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y}$ est réalisé par une stratégie pure de L). En déduire le théorème minimax à partir du théorème de dualité en programmation linéaire.

12.5 Notes

Nous recommandons la lecture du livre de Chvátal [52] pour une bonne introduction à la programmation linéaire. De très nombreux livres, par exemple, Dantzig [62], Karloff [168], Nemhauser et Wolsey [221] et Schrijver [246], explorent ce sujet. La programmation linéaire a été très utilisée en optimisation combinatoire, voir Ahuja, Magnanti et Orlin [3], Cook, Cunningham, Pulleyblank et Schrijver [55], Grötschel, Lovász et Schrijver [124], Lovász [202], Lovász et Plummer [203] et Papadimitriou et Steiglitz [226]. Reportez-vous à Lovász et Plummer [203] pour une bonne explication de l'algorithme d'Edmonds pour le couplage pondéré. Concernant la résolution d'un programme linéaire à base d'oracle séparateur, nous conseillons la lecture des livres de Grötschel, Lovász et Schrijver [123, 124] et de Schrijver [246].

13 Alignement dual pour la couverture par ensembles

Nous présentons dans ce chapitre, la méthode de l'alignement dual,[1] qui utilise la théorie de la dualité en programmation linéaire pour analyser les performances d'algorithmes combinatoires. Nous présentons cette méthode en proposant une autre analyse de l'algorithme glouton naturel (algorithme 2.2) pour la couverture par ensembles (problème 2.1). Nous découvrirons ici sur quel minorant est fondé l'algorithme — étude que nous avions différée en section 2.1. La puissance de cette méthode s'illustrera tout particulièrement par la facilité avec laquelle elle permet d'analyser diverses généralisations du problème de la couverture par ensembles (voir section 13.2).

Supposons que le problème étudié soit un problème de minimisation. La méthode de l'alignement dual peut se décrire ainsi. L'algorithme à analyser est de nature combinatoire — il s'agit de l'algorithme glouton pour la couverture par ensembles. En utilisant une relaxation d'un programme linéaire du problème et son dual, on commence par démontrer que le coût de la solution entière du primal construite par l'algorithme est totalement couvert par une solution du dual, (malheureusement) souvent irréalisable ; par *totalement couvert*,[2] nous signifions que la valeur objectif de la solution primale calculée est inférieure à la valeur objectif de la solution duale. L'étape principale de l'analyse consiste alors à diviser la solution duale par un facteur adéquat pour qu'elle devienne réalisable, c'est-à-dire pour qu'elle s'aligne sur une solution réalisable. Le coût de la solution duale réduite obtenue est alors un minorant de OPT. Le facteur de réduction, quant à lui, est un majorant du facteur d'approximation garanti par l'algorithme.

13.1 Analyse de l'algorithme glouton pour la couverture par ensembles par alignement dual

Commençons par encoder le problème de la couverture par ensembles sous forme d'un programme linéaire. Nous associons à chaque ensemble $S \in \mathcal{S}$, une variable x_S à valeurs dans $\{0, 1\}$. Cette variable vaudra 1 ssi l'ensemble S est sélectionné dans la couverture. La contrainte est clairement que tout élément $e \in U$ doit appartenir à au moins un des ensembles sélectionnés.

[1] *Dual fitting method*, en anglais.
[2] *Fully paid*, en anglais.

$$\begin{aligned} &\text{minimiser} && \sum_{S\in\mathcal{S}} c(S)x_S && && (13.1)\\ &\text{sous les contraintes} && \sum_{S\,:\,e\in S} x_S \geq 1, && e\in U\\ & && x_S \in \{0,1\}, && S\in\mathcal{S} \end{aligned}$$

On obtient la relaxation de ce programme en nombres entiers en autorisant x_S à prendre ses valeurs dans $[0,1]$. Mais, comme les bornes supérieures sur les x_S sont redondantes (du fait de la minimisation), nous obtenons le programme suivant. Les solutions de ce programme peuvent s'interpréter comme des couvertures par ensembles fractionnaires.

$$\begin{aligned} &\text{minimiser} && \sum_{S\in\mathcal{S}} c(S)x_S && && (13.2)\\ &\text{sous les contraintes} && \sum_{S\,:\,e\in S} x_S \geq 1, && e\in U\\ & && x_S \geq 0, && S\in\mathcal{S} \end{aligned}$$

Exemple 13.1 Voici un exemple où une couverture par ensembles fractionnaires peut coûter moins cher que la solution optimale entière. Prenons $U = \{e, f, g\}$ et pour ensembles $S_1 = \{e, f\}$, $S_2 = \{f, g\}$ et $S_3 = \{e, g\}$, tous de coût unitaire. Toute solution entière doit sélectionner deux de ces ensembles pour un coût total de 2, alors qu'en sélectionnant tous les ensembles avec une pondération de $1/2$, on obtient une couverture fractionnaire coûtant $3/2$. □

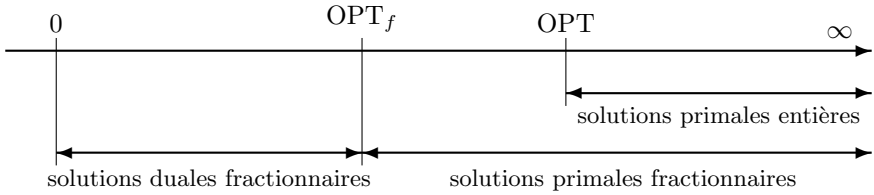
En associant une variable y_e à tout élément $e \in U$, nous obtenons le programme dual suivant.

$$\begin{aligned} &\text{maximiser} && \sum_{e\in U} y_e && && (13.3)\\ &\text{sous les contraintes} && \sum_{e\,:\,e\in S} y_e \leq c(S), && S\in\mathcal{S}\\ & && y_e \geq 0, && e\in U \end{aligned}$$

Il y a-t-il une raison intuitive pour laquelle (13.3) est le dual de (13.2) ? Existe-t-il une relation intuitive entre ces deux problèmes ? Par expérience, ce ne sont en général pas les bonnes questions. Comme nous l'avons vu section 12.1, le dual se construit de manière purement mécanique. En revanche, une fois le dual obtenu, on peut en rechercher une interprétation intuitive, parfois même combinatoire. Grâce à cette procédure mécanique, nous pouvons facilement obtenir le dual de problèmes très complexes, ce qui confère un champ d'application très large à la dualité en programmation linéaire.

Une interprétation intuitive du programme (13.3) est qu'il cherche à empaqueter des « choses » dans les éléments, en essayant de maximiser le volume total empaqueté, sous la contrainte qu'aucun ensemble ne « déborde ». Un ensemble *déborde* si le volume total empaqueté dans ses éléments dépasse son coût. Lorsque les coefficients de la matrice de contraintes, de la fonction objectif, et les termes constants sont tous positifs (ou nuls), le programme primal de minimisation est appelé *programme linéaire de couverture* et le programme dual de maximisation, *programme linéaire d'empaquetage*.[3] Les programmes (13.2) et (13.3) forment une paire de programmes linéaires couverture-empaquetage. Nous retrouverons souvent de telles paires dans les chapitres qui suivent.

Nous pouvons maintenant décrire le minorant sous-jacent à l'algorithme 2.2. Notons OPT_f le coût optimal d'une couverture par ensembles fractionnaire, c'est-à-dire d'une solution optimale du programme (13.2). Clairement, $\mathrm{OPT}_f \leqslant \mathrm{OPT}$, le coût d'une couverture optimale (entière). Le coût de toute solution réalisable du programme dual (13.3) minore OPT_f et donc OPT. L'algorithme 2.2 utilise implicitement ce minorant.



L'algorithme 2.2 associe des variables duales $\mathrm{prix}(e)$ à chaque élément e. Remarquons que le coût de la couverture sélectionnée par l'algorithme est totalement couvert par cette solution duale. Cette solution duale n'est pas réalisable en général (voir exercice 13.2). Mais nous allons démontrer qu'en réduisant cette solution duale d'un facteur H_n, elle s'aligne alors sur une solution réalisable, c'est-à-dire que plus aucun ensemble ne déborde. Posons, pour tout élément e,

$$y_e = \frac{\mathrm{prix}(e)}{H_n}.$$

La solution duale réalisable $\boldsymbol{y}$ est utilisée implicitement par l'algorithme 2.2 pour minorer OPT.

Lemme 13.2 *Le vecteur $\boldsymbol{y}$ défini ci-dessus est une solution réalisable du programme dual (13.3).*

Preuve : Nous devons démontrer qu'aucun ensemble ne déborde pour la solution $\boldsymbol{y}$. Étudions un ensemble $S \in \mathcal{S}$ à k éléments. Numérotons ses éléments

[3] *Covering LP* et *packing LP*, en anglais.

$e_1, \ldots, e_k$ dans l'ordre dans lequel ils ont été couverts par l'algorithme (arbitrairement en cas d'*ex æquo*).

Étudions l'itération où l'algorithme couvre l'élément e_i. À cet instant, S contient au moins $k-i+1$ éléments non couverts. Ainsi, S peut couvrir e_i à un coût moyen inférieur à $c(S)/(k-i+1)$. Puisque l'algorithme choisit l'ensemble de coût efficace minimum durant cette itération, $\text{prix}(e_i) \leqslant c(S)/(k-i+1)$. Par conséquent,

$$y_{e_i} \leq \frac{1}{H_n} \cdot \frac{c(S)}{k-i+1}.$$

Ainsi, en sommant sur tous les éléments de S,

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \right) = \frac{H_k}{H_n} \cdot c(S) \leq c(S).$$

S ne déborde donc pas pour la solution $\boldsymbol{y}$. □

Théorème 13.3 *L'algorithme glouton pour la couverture par ensemble est une H_n-approximation.*

Preuve : Le coût de la couverture sélectionnée est

$$\sum_{e \in U} \text{prix}(e) = H_n \left(\sum_{e \in U} y_e \right) \leqslant H_n \cdot \text{OPT}_f \leqslant H_n \cdot \text{OPT},$$

où OPT_f désigne le coût optimal d'une couverture fractionnaire. L'avant-dernière inégalité provient du fait que $\boldsymbol{y}$ est une solution réalisable du dual. □

13.1.1 Le facteur d'approximation peut-il être amélioré ?

Revenons sur les trois questions posées à la section 1.1.2, concernant l'amélioration du facteur d'approximation pour la couverture par sommets, dans le cadre de la couverture par ensembles. Nous avons déjà répondu à la première et à la troisième dans la section 2.1.

Un corollaire du théorème 13.3 est que H_n majore le saut intégral de la relaxation (13.2). L'exemple 13.4 démontrera que cette borne est asymptotiquement juste. Comme le saut intégral d'une relaxation d'un programme linéaire est moralement le meilleur facteur d'approximation qu'on puisse espérer en utilisant cette relaxation, la réponse à la seconde question est « non ».

Exemple 13.4 Étudions l'instance suivante de la couverture par ensembles. Soit $k > 0$ un entier, posons $n = 2^k - 1$, et $U = \{e_1, e_2, \ldots, e_n\}$. Considérons

l'écriture binaire à k bits de chaque entier i, $1 \leqslant i \leqslant n$, vue comme un vecteur $\mathbf{i}$ de dimension k sur GF[2]. Posons $S_i = \{e_j : \mathbf{i} \cdot \mathbf{j} = 1\}$, pour tout $1 \leqslant i \leqslant n$, où $\mathbf{i} \cdot \mathbf{j}$ désigne le produit scalaire des deux vecteurs. Prenons enfin $\mathcal{S} = \{S_1, \ldots, S_n\}$, où tous ces ensembles ont un coût unitaire.

On vérifie facilement que chaque ensemble contient exactement $2^{k-1} = (n+1)/2$ éléments et que chaque élément appartient exactement à $(n+1)/2$ ensembles. Ainsi, $x_i = 2/(n+1)$, $1 \leqslant i \leqslant n$, définit une couverture fractionnaire par ensembles coûtant $2n/(n+1)$.

Nous allons maintenant démontrer que toute couverture entière doit sélectionner au moins k des ensembles. Étudions une union quelconque de p des ensembles, avec $p < k$. Notons $i_1, \ldots, i_p$ les indices associés à ces ensembles et $\boldsymbol{A}$, la matrice $p \times k$ sur GF[2] dont les colonnes sont les vecteurs $\mathbf{i}_1, \ldots, \mathbf{i}_p$. Puisque le rang de A est $< k$, son noyau est de dimension $\geqslant 1$, et contient donc un vecteur $\mathbf{j}$ non nul. Puisque $\boldsymbol{A}\mathbf{j} = \mathbf{0}$, l'élément e_j n'appartient à aucun des p ensembles. Ces p ensembles ne couvrent donc pas U.

Nous en concluons que toute couverture par ensemble (entière) coûte au moins $k = \log_2(n+1)$. Nous obtenons donc le minorant suivant du saut intégral pour cette instance :

$$\left(\frac{n+1}{2n}\right) \cdot \log_2(n+1) > \frac{\log_2 n}{2}.$$

□

13.2 Variantes de la couverture par ensembles

L'algorithme glouton et son analyse par alignement dual s'étendent naturellement à diverses généralisations[4] du problème de la couverture par ensembles (voir exercice 13.4).

- **Multicouverture par ensembles :** Un nombre de couverture r_e est associé à chaque élément e. Le but est de couvrir, pour un coût minimum, chaque élément e au moins r_e fois. On suppose que sélectionner k fois un ensemble S coûte $k \cdot c(S)$.
- **Multicouverture par des multi-ensembles :** L'entrée est un ensemble de multi-ensembles d'éléments de U. Un *multi-ensemble* contient un certain nombre de copies de chaque élément. La multiplicité $M(S, e)$ de e dans S est contrainte à vérifier que $\forall S\ \forall e\ M(S, e) \leqslant r_e$. Le but est le même que précédemment.
- **Programmes de couverture entiers :** Ce sont les programmes linéaires de la forme :

[4] Respectivement, *set multicover*, *multiset multicover* et *covering integer programs*, en anglais.

minimiser $\boldsymbol{c} \cdot \boldsymbol{x}$

sous la contrainte $A\boldsymbol{x} \geqslant \boldsymbol{b}$,

où les coefficients de $\boldsymbol{A}$, $\boldsymbol{b}$, et $\boldsymbol{c}$ sont positifs, et $\boldsymbol{x}$ est contraint à prendre des valeurs entières positives.

13.2.1 Alignement dual et multicouverture contrainte par ensembles

Nous présentons dans cette section une H_n-approximation pour le problème de la *multicouverture par ensembles contrainte*,[5] c'est-à-dire où on n'autorise pas à sélectionner un ensemble plusieurs fois. Une particularité de ce problème est que sa relaxation linéaire et son dual ont des coefficients négatifs et donc ne forment pas une paire de programmes couverture-empaquetage.

Soit $r_e \in \mathbf{N}$, le nombre de couverture requis pour chaque élément $e \in U$. Le programme linéaire en nombres entiers pour la multicouverture contrainte n'est pas très différent de celui de la couverture simple.

$$\begin{aligned} &\text{minimiser} && \sum_{S \in \mathcal{S}} c(S) x_S && && (13.4)\\ &\text{sous les contraintes} && \sum_{S\,:\,e \in S} x_S \geq r_e, && e \in U \\ & && x_S \in \{0,1\}, && S \in \mathcal{S} \end{aligned}$$

Remarquons cependant que, dans la relaxation, les contraintes $x_S \leq 1$ ne sont plus redondantes. En les éliminant, on risque de sélectionner plusieurs fois le même ensemble. La relaxation (13.5) est donc différente de celle de la couverture par ensembles. En particulier, du fait des coefficients négatifs dans la matrice de contraintes et dans les termes constants, la relaxation n'est même pas un programme de couverture. La méthode d'analyse s'étend malgré tout à cette nouvelle généralisation.

$$\begin{aligned} &\text{minimiser} && \sum_{S \in \mathcal{S}} c(S) x_S && && (13.5)\\ &\text{sous les contraintes} && \sum_{S\,:\,e \in S} x_S \geq r_e, && e \in U \\ & && -x_S \geq -1, && S \in \mathcal{S} \\ & && x_S \geq 0, && S \in \mathcal{S} \end{aligned}$$

[5] *Constrained set multicover problem*, en anglais.

Associons une nouvelle variable z_S à chaque nouvelle contrainte du primal. Le programme dual a lui aussi des coefficients négatifs dans sa matrice de contraintes et n'est donc pas non plus un programme d'empaquetage. En particulier, on peut compenser le débordement en y_e d'un ensemble S en augmentant z_S et obtenir une solution réalisable (dont la valeur objectif est diminuée d'autant). Ainsi, il est souhaitable de faire déborder les ensembles, puisque les y_e apparaissent pondérés par les r_e dans la fonction objectif.

$$\begin{aligned} &\text{maximiser} && \sum_{e\in U} r_e y_e - \sum_{S\in\mathcal{S}} z_S && && (13.6)\\ &\text{sous les contraintes} && \left(\sum_{e\,:\,e\in S} y_e\right) - z_S \le c(S), && S\in\mathcal{S} \\ & && y_e \ge 0, && e\in U \\ & && z_S \ge 0, && S\in\mathcal{S} \end{aligned}$$

Voici un algorithme glouton. Nous dirons qu'un élément e est *actif*, s'il est couvert par strictement moins de r_e des ensembles sélectionnés. Durant chaque itération, l'algorithme choisit l'ensemble non sélectionné le plus rentable, c'est-à-dire celui dont le *coût efficace* (le coût rapporté au nombre d'éléments actifs couverts) est maximum. L'algorithme s'arrête lorsqu'il ne reste plus d'éléments actifs, c'est-à-dire quand tous les éléments sont couverts au moins autant de fois que leur nombre de couverture.

Lorsqu'un ensemble S est choisi, son coût est réparti uniformément sur les éléments actifs qu'il couvre, comme suit : si S couvre e pour la j-ième fois, $\text{prix}(e,j)$ est fixé au coût efficace de S. Comme le coût efficace des ensembles sélectionnés croît durant l'exécution de l'algorithme, pour tout élément e, $\text{prix}(e,1) \leqslant \text{prix}(e,2) \leqslant \ldots \leqslant \text{prix}(e,r_e)$.

À la fin de l'algorithme, nous définissons la solution duale associée comme suit. Posons pour tout $e \in U$, $\alpha_e = \text{prix}(e,r_e)$ et pour tout $S \in \mathcal{S}$ sélectionné par l'algorithme,

$$\beta_S = \sum_{e \text{ couvert par } S} \left(\text{prix}(e,r_e) - \text{prix}(e,j_e)\right),$$

où j_e est le numéro de la copie de e, couverte par S. Remarquons que β_S est positif, car $\text{prix}(e,j_e) \leqslant \text{prix}(e,r_e)$. Si S n'est pas sélectionné par l'algorithme, nous posons $\beta_S = 0$.

Lemme 13.5 *Le coût de la multicouverture par ensembles construite par l'algorithme est totalement couvert par la solution duale* $(\boldsymbol{\alpha},\boldsymbol{\beta})$.

Preuve : Puisque le coût des ensembles sélectionnés par l'algorithme a été réparti sur les éléments couverts, le coût total de la couverture engendrée est :

$$\sum_{e \in U} \sum_{j=1}^{r_e} \text{prix}(e, j).$$

Or, la valeur objectif de la solution duale $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ est

$$\sum_{e \in U} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = \sum_{e \in U} \sum_{j=1}^{r_e} \text{prix}(e, j).$$

CQFD. □

La solution duale proposée ci-dessus n'est pas réalisable en général. Nous allons démontrer toutefois qu'elle le devient en la réduisant d'un facteur H_n. Pour tout élément $e \in U$ et tout ensemble $S \in \mathcal{S}$, posons :

$$y_e = \frac{\alpha_e}{H_n} \quad \text{et} \quad z_S = \frac{\beta_S}{H_n}.$$

Lemme 13.6 $(\boldsymbol{y}, \boldsymbol{z})$ *est une solution réalisable du programme dual (13.6).*

Preuve : Considérons un ensemble $S \in \mathcal{S}$ de taille k. Numérotons ses éléments $e_1, \ldots, e_k$ dans l'ordre dans lequel leurs nombres de couverture ont été atteints, c'est-à-dire par ordre de fin d'activité.

Supposons tout d'abord que S n'a pas été sélectionné par l'algorithme. Lorsque l'algorithme s'apprête à couvrir la dernière copie de e_i, S contient au moins $k - i + 1$ éléments actifs. Ainsi,

$$\text{prix}(e_i, r_{e_i}) \leq \frac{c(S)}{k - i + 1}.$$

Or, $z_S = 0$ donc

$$\begin{aligned} \left(\sum_{i=1}^{k} y_{e_i} \right) - z_S &= \frac{1}{H_n} \sum_{i=1}^{k} \text{prix}(e_i, r_{e_i}) \\ &\leq \frac{c(S)}{H_n} \cdot \left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \right) \leq c(S) \ . \end{aligned}$$

Supposons maintenant que S soit choisi par l'algorithme et que, juste avant sa sélection, $k' \geq 0$ éléments de S soient déjà totalement couverts. Alors,

$$\left(\sum_{i=1}^{k} y_{e_i}\right) - z_S$$
$$= \frac{1}{H_n} \cdot \left[\sum_{i=1}^{k} \text{prix}(e_i, r_{e_i}) - \sum_{i=k'+1}^{k} (\text{prix}(e_i, r_{e_i}) - \text{prix}(e_i, j_i))\right]$$
$$= \frac{1}{H_n} \cdot \left[\sum_{i=1}^{k'} \text{prix}(e_i, r_{e_i}) + \sum_{i=k'+1}^{k} \text{prix}(e_i, j_i)\right] ,$$

où j_i est le numéro de la copie de e_i couverte par S, pour $i \in \{k'+1, \ldots, k\}$. Comme le coût de S est réparti uniformément sur les copies qu'il couvre, $\sum_{i=k'+1}^{k} \text{prix}(e_i, j_i) = c(S)$. Enfin, étudions les e_i avec $i \in \{1, \ldots, k'\}$. Lorsque la dernière copie de e_i est sur le point d'être couverte, S n'est pas encore sélectionné, et couvre au moins $k - i + 1$ éléments actifs. Ainsi, $\text{prix}(e_i, r_{e_i}) \leq c(S)/(k - i + 1)$. Par conséquent,

$$\left(\sum_{i=1}^{k} y_{e_i}\right) - z_S \leq \frac{c(S)}{H_n} \cdot \left(\frac{1}{k} + \cdots + \frac{1}{k - k' + 1} + 1\right) \leq c(S).$$

$(\boldsymbol{y}, \boldsymbol{z})$ est donc bien une solution réalisable du dual. □

Théorème 13.7 *L'algorithme glouton est une H_n-approximation pour le problème de la multicouverture par ensembles contrainte.*

Preuve : D'après les lemmes 13.5 et 13.6, le coût total de la multicouverture construite par l'algorithme vaut :

$$\sum_{e \in U} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = H_n \cdot \left[\sum_{e \in U} r_e y_e - \sum_{S \in \mathcal{S}} z_S\right] \leqslant H_n \cdot \text{OPT} .$$

□

Un corollaire du théorème 13.7 est que le saut intégral de la relaxation (13.5) est majoré par H_n. La situation est très différente pour la multicouverture par multi-ensembles, contrainte à ne sélectionner chaque multi-ensemble qu'au plus une fois, car son saut intégral n'est majoré par aucune fonction de n (voir exercice 13.5).

13.3 Exercices

13.1 Démontrez que l'analyse par alignement dual des algorithmes gloutons pour la couverture par ensembles et la multicouverture par ensembles

contrainte donne en fait un facteur d'approximation H_k, où k est la taille du plus grand ensemble de l'instance. Remarquez la facilité avec laquelle ce résultat est obtenu par dualité (comparez avec l'exercice 2.8).

13.2 Exhibez un exemple pour lequel la solution duale $(\text{prix}(e))_{e\in U}$ calculée par l'algorithme 2.2 fait déborder certains ensembles S — d'un facteur $H_{|S|}$ asymptotiquement.

13.3 Exhibez des exemples où la minoration $\boldsymbol{y}$ utilisée par l'algorithme 2.2 est à un facteur $\Omega(\log n)$ de OPT.

13.4 Trouvez des algorithmes d'approximation pour les problèmes suivants.

1. Une H_n-approximation pour la multicouverture par ensembles.
2. Une H_m-approximation pour la multicouverture par multi-ensembles, où m est la taille du plus grand multi-ensemble de l'instance (les éléments sont comptés avec leurs multiplicité dans la taille d'un multi-ensemble).
3. Une $O(\log n)$-approximation pour les programmes de couverture entiers.

Indication : Pour la H_m-approximation de la multicouverture par multi-ensembles, fixez les variables duales en fonction du coût moyen des éléments couverts, c'est-à-dire

$$y_e = \frac{1}{H_m} \sum_{i=1}^{r_e} \text{prix}(e,i)/r_e.$$

Puis utilisez les méthodes de changement d'échelle[6] et d'arrondi pour réduire les programmes de couverture entiers à une multicouverture par multi-ensembles, où m est borné par un polynôme en n, au prix d'une petite erreur (que vous inclurez dans le facteur d'approximation).

13.5 Démontrez que le saut intégral de la relaxation pour les deux variantes suivantes de la multicouverture par multi-ensembles, inspirée du programme (13.2), n'est borné par aucune fonction de n.

1. Lever la contrainte sur les instances : $\forall e\ \forall S\ M(S,e) \leqslant r_e$.
2. Imposer la contrainte qu'aucun multi-ensemble ne soit sélectionné plusieurs fois.

Quel est le meilleur facteur d'approximation que vous puissiez garantir pour l'algorithme glouton dans la seconde variante ? Pourquoi la preuve du facteur H_n de la section 13.2 ne fonctionne-t-elle pas ici ?

13.6 (Mihail [215]) Étudions la variante suivante du problème de la multicouverture par ensembles. Soient U un ensemble univers de taille n, et $\mathcal{S}$

[6] *Scaling*, en anglais.

une collection de sous-ensembles de U. Le coût de chaque ensemble $S \in \mathcal{S}$ est défini comme une fonction croissante du temps $t \in \{1, \ldots, T\}$. À chaque élément de U est attribué un nombre de couverture, également croissant en fonction du temps. Le problème est de sélectionner des ensembles de coût total minimum de façon à satisfaire les nombres de recouvrement requis par les éléments à tout instant. Chaque ensemble peut être sélectionné plusieurs fois ; le coût d'un ensemble est fixé au moment de sa sélection. Une fois sélectionné, l'ensemble reste dans la couverture sans rien coûter aux étapes suivantes.

Proposez une H_n-approximation pour ce problème (une $H_{(n \cdot T)}$-approximation est évidente).

13.7 Dans de nombreuses situations réalistes, le coût d'une sélection multiple ne croit pas linéairement avec la multiplicité. Cette fonction est souvent seulement concave. La variante suivante de la multicouverture par ensembles capture cette situation. Le coût de la sélection de chaque ensemble S_i est défini par une fonction concave f_i du nombre de sélections. Le problème est toujours de satisfaire toutes les contraintes de recouvrement pour un coût minimum. Proposez une H_n-approximation pour ce problème.
Indication : Réduire le problème à un problème de multicouverture par multi-ensembles. Associez à chaque ensemble S_i, des ensembles S_i^j, $j \geqslant 1$, où S_i^j contient j copies de chaque élément de S_i et coûte $f_i(j)$. L'algorithme glouton sur cette instance donne le bon facteur d'approximation. Puis démontrez qu'il n'est pas nécessaire de construire tous les S_i^j. À chaque itération de l'algorithme glouton, le multi-ensemble le plus rentable se calcule directement en temps polynomial, même si le nombre de conditions est exponentiel.

13.4 Notes

L'analyse par alignement dual de l'algorithme de couverture par ensembles est due à Lovász [200] et Chvátal [51]. Rajagopalan et Vazirani [236] sont à l'origine de l'analyse de la multicouverture par ensembles. Pour des algorithmes de résolution des programmes de couverture entiers, se reporter à Dobson [66] et Rajagopalan et Vazirani [236].

14 Arrondi en programmation linéaire et couverture par ensembles

Nous introduisons ici la technique d'arrondi en programmation linéaire et développons deux algorithmes d'approximation pour le problème de la couverture par ensembles (problème 2.1). Le premier, fondé sur un schéma d'arrondi élémentaire, est une f-approximation, où f est la fréquence maximale d'un élément. Le second algorithme est une $O(\log n)$-approximation, qui illustrera l'utilisation de la randomisation dans les procédures d'arrondi.

Considérons le polyèdre défini par les solutions réalisables d'une relaxation d'un programmation linéaire. Pour certains problèmes, les solutions extrémales de ce polyèdre ont des propriétés intéressantes qui permettent de construire des algorithmes à base d'arrondi. Par exemple, les solutions extrémales peuvent être *demi-entières*, c'est-à-dire à coordonnées dans $\{0, 1/2, 1\}$. Nous verrons, section 14.3, que la couverture par sommets a cette propriété remarquable. Il en découle une 2-approximation immédiate : il suffit d'arrondir à 1 toutes les variables valant 1/2. Nous verrons au chapitre 23 une autre propriété plus générale qui permet une technique d'arrondi plus performante (l'arrondi itératif).

14.1 Un algorithme d'arrondi simple

Reprenons la relaxation linéaire (13.2) du problème de la couverture par ensembles. Une première façon de convertir une solution de la relaxation en une solution entière est d'arrondir toutes les variables non nulles à 1. Il est facile d'exhiber des exemples où le coût de la solution est multiplié par un facteur $\Omega(n)$ *via* ce procédé (voir exemple 14.3). Néanmoins, cet algorithme très simple réalise l'approximation désirée, d'un facteur f (voir exercice 14.1). Nous étudierons plutôt une variante très proche de cet algorithme, qui sera plus facile à analyser, et qui sélectionne moins d'ensembles en général :

Algorithme 14.1 (Couverture par ensembles par arrondi)

1. Calculer une solution optimale de la relaxation du programme linéaire.
2. Sélectionner tous les ensembles S tels que $x_S \geqslant 1/f$ dans cette solution.

Théorème 14.2 *L'algorithme 14.1 est une f-approximation pour le problème de la couverture par ensembles.*

Preuve : Notons $\mathcal{C}$ l'ensemble des ensembles sélectionnés. Prenons un élément e arbitraire. Puisque e appartient à moins de f ensembles et que $\sum_{S \ni e} x_S \geqslant 1$, il existe un de ces ensembles dont la valeur attribuée par la couverture fractionnaire est $\geqslant 1/f$. $\mathcal{C}$ couvre donc bien e. C'est donc une couverture valide. La phase d'arrondi multiplie chaque x_S par f au plus. $\mathcal{C}$ coûte donc moins que f fois le coût de la couverture fractionnaire. CQFD. □

Pour les instances de couverture par ensembles issues des problèmes de couverture par sommets, nous avons $f = 2$. L'algorithme 14.1 donne donc une 2-approximation pour le problème de la couverture pondérée par sommets. Nous retrouvons donc la même garantie que pour la version non pondérée (théorème 2.7).

Exemple 14.3 Voici une instance critique pour l'algorithme 14.1. Pour simplifier, représentons les instances de la couverture par ensembles par des hypergraphes : les sommets représentent les ensembles et les hyper-arêtes les éléments (c'est une généralisation de la réduction entre la couverture par sommets et les instances de fréquence 2 de la couverture par ensembles). La relation d'incidence dans l'hypergraphe correspond à la relation d'inclusion dans la couverture.

Prenons k ensembles disjoints $V_1, \ldots, V_k$ de taille n. Notre hypergraphe a pour ensemble de sommets $V = V_1 \cup \cdots \cup V_k$ et n^k arêtes ; chaque hyper-arête relie ensemble k sommets $v_1, \ldots, v_k$ choisis respectivement dans $V_1, \ldots, V_k$. Chaque ensemble coûte 1. La sélection de chaque ensemble avec un cœfficient $1/k$ définit donc une couverture fractionnaire optimale, coûtant n. À partir de cette solution fractionnaire, l'algorithme d'arrondi sélectionne les nk ensembles, alors que la sélection des n ensembles correspondant aux sommets de V_1 donne une couverture par ensembles coûtant n. □

14.2 Arrondi randomisé

Une idée naturelle pour arrondir une solution optimale fractionnaire est d'interpréter les fractions comme des probabilités, et de tirer à pile-ou-face suivant ces probabilités pour arrondir la solution. Nous allons démontrer que cette idée conduit à une $O(\log n)$-approximation randomisée pour le problème de la couverture par ensembles.

Nous commencerons par démontrer que chaque élément est couvert avec une probabilité constante par les ensembles sélectionnés. En répétant le processus $O(\log n)$ fois puis en conservant tous les ensembles qui ont été sélectionnés au moins une fois, nous obtiendrons une couverture par ensembles avec forte probabilité par un argument standard de collectionneur

de coupons.[1] L'espérance du coût de la couverture ainsi construite sera $O(\log n) \cdot \mathrm{OPT}_f \leqslant O(\log n) \cdot \mathrm{OPT}$ (où OPT_f désigne le coût d'une solution optimale de la relaxation du programme linéaire). L'inégalité de Markov assurera que ce résultat est obtenu avec forte probabilité. Voyons maintenant la preuve en détail.

Soit $\boldsymbol{x} = \boldsymbol{p}$, une solution optimale de la relaxation linéaire. Chaque $S \in \mathcal{S}$ est sélectionné avec probabilité p_S, la variable associée à S dans $\boldsymbol{p}$. Notons $\mathcal{C}$, l'ensemble des ensembles sélectionnés. L'espérance du coût de $\mathcal{C}$ est

$$\mathbf{E}[c(\mathcal{C})] = \sum_{S \in \mathcal{S}} \mathbf{Pr}[S \text{ est sélectionné}] \cdot c(S) = \sum_{S \in \mathcal{S}} p_S \cdot c(S) = \mathrm{OPT}_f .$$

Calculons maintenant la probabilité qu'un élément $a \in U$ soit couvert par $\mathcal{C}$. Supposons que a appartient à k des ensembles de $\mathcal{S}$. Notons $p_1, \ldots, p_k$ les probabilités associées à ces ensembles. Comme a est couvert de façon fractionnaire par la solution optimale $\boldsymbol{p}$, $p_1 + p_2 + \cdots + p_k \geqslant 1$. Un calcul élémentaire donne que sous cette condition, la probabilité que a soit couvert par C est minimale si tous les p_i sont égaux à $1/k$. Donc,

$$\mathbf{Pr}[a \text{ est couvert par } \mathcal{C}] \geqslant 1 - \left(1 - \frac{1}{k}\right)^k \geqslant 1 - \frac{1}{e},$$

où e est la base du logarithme naturel. Ainsi, tout élément est couvert par $\mathcal{C}$ avec une probabilité constante.

Pour obtenir une couverture par ensembles complète, choisissons indépendamment $d \log n$ « sous-couvertures » comme précédemment, et prenons-en l'union, $\mathcal{C}'$. Choisissons la constante d de sorte que

$$\left(\frac{1}{e}\right)^{d \log n} \leqslant \frac{1}{4n}.$$

Alors,

$$\mathbf{Pr}[a \text{ n'est pas couvert par } \mathcal{C}'] \leqslant \left(\frac{1}{e}\right)^{d \log n} \leqslant \frac{1}{4n}.$$

En sommant sur tous les éléments $a \in U$, nous en concluons que

$$\mathbf{Pr}[\mathcal{C}' \text{ n'est pas une couverture par ensembles valide}] \leqslant n \cdot \frac{1}{4n} \leqslant \frac{1}{4}.$$

Clairement, $\mathbf{E}[c(\mathcal{C}')] \leqslant \mathrm{OPT}_f \cdot d \log n$. L'inégalité de Markov (voir section B.2) avec $t = \mathrm{OPT}_f \cdot 4d \log n$, donne que

$$\mathbf{Pr}[c(\mathcal{C}') \geqslant \mathrm{OPT}_f \cdot 4d \log n] \leqslant \frac{1}{4}.$$

[1] *Coupon collector*, en anglais.

La probabilité de l'union des deux événements indésirables est donc $\leqslant 1/2$. Ainsi,

$$\mathbf{Pr}\left[\begin{array}{l}\mathcal{C}' \text{ est une couverture par ensembles valide} \\ \text{coûtant} \leqslant \mathrm{OPT}_f \cdot 4d \log n\end{array}\right] \geqslant \frac{1}{2}.$$

Remarquons que nous pouvons tester en temps polynomial si $\mathcal{C}'$ vérifie bien les deux conditions. Dans le cas contraire, il suffit de relancer l'algorithme une nouvelle fois. L'espérance du nombre nécessaire d'exécutions est inférieur à 2.

14.3 Solutions demi-entières pour la couverture par sommets

Étudions le problème de la couverture par sommets avec des poids arbitraires $c : V \to \mathbf{Q}^+$ sur les sommets. Un programme linéaire en nombres entiers encodant ce problème est :

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{v \in V} c(v) x_v & \\ \text{sous les contraintes} & x_u + x_v \geq 1, & uv \in E \\ & x_v \in \{0,1\}, & v \in V \end{array} \tag{14.1}$$

et sa relaxation :

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{v \in V} c(v) x_v & \\ \text{sous les contraintes} & x_u + x_v \geq 1, & uv \in E \\ & x_v \geq 0, & v \in V \end{array} \tag{14.2}$$

Rappelons qu'une *solution extrémale*[2] pour un ensemble de contraintes linéaires est une solution réalisable qui ne peut pas s'écrire comme une combinaison convexe de deux solutions réalisables distinctes. Une solution du programme (14.2) est dite *demi-entière* si ses variables sont à valeurs dans $\{0, 1/2, 1\}$.

Lemme 14.4 *Toute solution extrémale de (14.2) est demi-entière.*

Preuve : Soit $\boldsymbol{x}$ une solution réalisable du programme (14.2) qui n'est pas demi-entière. Nous allons démontrer que $\boldsymbol{x}$ est la combinaison convexe de deux solutions réalisables distinctes, et qu'elle n'est donc pas extrémale.

[2] *Extreme point solution*, en anglais.

Étudions l'ensemble des sommets auxquels $\boldsymbol{x}$ n'attribue pas de valeurs demi-entières. Partitionnons cet ensemble de la façon suivante.

$$V_+ = \left\{ v \,:\, \frac{1}{2} < x_v < 1 \right\}, \quad V_- = \left\{ v \,:\, 0 < x_v < \frac{1}{2} \right\}.$$

Définissons les deux solutions suivantes, fonctions d'un $\varepsilon > 0$.

$$y_v = \begin{cases} x_v + \varepsilon, & \text{si } v \in V_+ \\ x_v - \varepsilon, & \text{si } v \in V_- \\ x_v, & \text{sinon} \end{cases} \quad \text{et} \quad z_v = \begin{cases} x_v - \varepsilon, & \text{si } v \in V_+ \\ x_v + \varepsilon, & \text{si } v \in V_- \\ x_v, & \text{sinon.} \end{cases}$$

Par hypothèse, $V_+ \cup V_- \neq \varnothing$ et donc $\boldsymbol{x}$ est distinct de $\boldsymbol{y}$ et de $\boldsymbol{z}$. De plus, $\boldsymbol{x}$ est la combinaison linéaire convexe de $\boldsymbol{y}$ et $\boldsymbol{z}$, car $\boldsymbol{x} = \frac{1}{2}(\boldsymbol{y} + \boldsymbol{z})$. Il ne nous reste plus qu'à démontrer pour conclure que, pour $\varepsilon > 0$ suffisamment petit, $\boldsymbol{y}$ et $\boldsymbol{z}$ sont des solutions réalisables du programme (14.2).

Il est facile d'assurer que toutes les coordonnées de $\boldsymbol{y}$ et $\boldsymbol{z}$ soient positives. Étudions à présent les contraintes liées aux arêtes. Considérons une arête uv et supposons que $x_u + x_v > 1$. Clairement, en choisissant ε suffisamment petit, nous assurons que ni $\boldsymbol{y}$, ni $\boldsymbol{z}$ ne violent cette contrainte. Supposons maintenant que $x_u + x_v = 1$. Il n'y a que trois possibilités pour x_u et x_v : $x_u = x_v = \frac{1}{2}$; $x_u = 0$ et $x_v = 1$; ou $u \in V_+$ et $v \in V_-$. Dans les trois cas, et pour tout choix de ε,

$$x_u + x_v = y_u + y_v = z_u + z_v = 1.$$

CQFD. □

Théorème 14.5 *Toute solution optimale et extrémale du programme (14.2) est demi-entière.*

Le théorème 14.5 donne directement une 2-approximation pour trouver une couverture par sommets pondérée : calculer une solution extrémale optimale, puis sélectionner les sommets qui se voient attribuer les valeurs 1/2 ou 1 par cette solution.

14.4 Exercices

14.1 Modifiez l'algorithme 14.1 pour qu'il sélectionne tous les ensembles de valeurs non nulles dans la solution fractionnaire. Démontrez que c'est également une f-approximation.
Indication : Utilisez le théorème des écarts complémentaires.

14.2 Considérez l'ensemble $\mathcal{C}$ des ensembles sélectionnés par l'algorithme d'arrondi randomisé. Démontrez qu'avec une probabilité constante, $\mathcal{C}$ couvre au moins la moitié des éléments pour un coût $O(\mathrm{OPT})$.

14.3 Donnez une $O(\log n)$-approximation randomisée à base d'arrondi pour chacun des problèmes suivants : la multicouverture par ensembles et la multicouverture par multi-ensembles (voir section 13.2).

14.4 Exhibez une instance critique qui ne soit pas un graphe biparti, pour l'algorithme à base de solutions demi-entières pour la couverture par sommets pondérée.

14.5 (J. Cheriyan) Donnez un algorithme polynomial pour le problème suivant. Étant donné un graphe G avec des poids positifs sur les sommets et une k-coloration valide de G (pas nécessairement optimale), trouver une couverture par sommets de poids $\leqslant (2 - \frac{2}{k})\,\mathrm{OPT}$, où k est le nombre de couleurs utilisées.

14.6 Exhibez un contre-exemple à l'affirmation suivante : « une instance de la couverture par ensembles pour laquelle tout élément appartient à exactement f ensembles, admet une solution fractionnaire optimale $(1/f)$-entière (c'est-à-dire dont les valeurs sont des multiples entiers de $1/f$) ».

14.7 Cet exercice présente un algorithme purement combinatoire pour trouver une couverture par sommets demi-entière optimale. Étant donné un graphe non orienté $G = (V, E)$ muni d'une fonction de coût positive c sur les sommets, construire un graphe biparti $H(V', V'', E')$ de la façon suivante. À chaque sommet $v \in V$ sont associés deux sommets $v' \in V'$ et $v'' \in V''$, de coût $c(v)/2$ chacun. À chaque arête $uv \in E$ sont associées les deux arêtes $u'v''$ et $u''v' \in E'$. Démontrez qu'il existe une bijection entre les couvertures par sommets de H et les couvertures par sommets demi-entières de G, qui préserve le coût total des couvertures. Puis, utilisez le fait qu'on peut calculer en temps polynomial une couverture par sommets optimale dans un graphe biparti, pour obtenir une couverture par sommets demi-entière optimale de G.

14.8 Étudions le programme (12.8) (voir exercice 12.8) sur un graphe $G = (V, E)$ qui n'est pas biparti.

1. Montrez que ce n'est pas une relaxation exacte du problème du couplage maximum dans G.
2. Démontrez que ce programme admet toujours une solution optimale demi-entière.

14.9 Afin de réduire le temps de calcul de l'algorithme obtenu dans l'exercice 9.7 pour l'empaquetage, essayons d'appliquer la méthode d'arrondi à la relaxation du programme en nombres entiers. Quel facteur d'approximation pouvez-vous garantir pour l'empaquetage par cette méthode ?

14.5 Notes

L'algorithme 14.1 est dû à Hochbaum [133]. Se référer à Srinivasan [253] pour une application plus élaborée de l'arrondi randomisé à la couverture par ensembles. Le théorème 14.5 a été démontré par Nemhauser et Trotter [222].

15 Schéma primal-dual et couverture par ensembles

Le schéma primal-dual est une méthode de choix qui permet d'obtenir des algorithmes combinatoires performants, tant en termes de facteur d'approximation qu'en temps de calcul. Ce chapitre commence par présenter la méthode, puis l'applique pour construire une f-approximation simple pour la couverture par ensembles (où f est la fréquence maximale d'un élément).

Le schéma primal-dual a ses origines en algorithmique exacte. Il a produit les solutions les plus performantes pour plusieurs problèmes fondamentaux de **P**, dont les problèmes de couplage, de flots et de plus courts chemins. Ces problèmes ont pour particularité que leur relaxation admet toujours des solutions optimales entières. Le théorème 12.3 nous dit que les solutions optimales d'un programme linéaire sont caractérisées par les conditions des écarts complémentaires. Le principe du schéma primal-dual en algorithmique exacte repose sur ces conditions. Partant de deux solutions, une du primal et une du dual, il cherche à satisfaire une à une les conditions des écarts complémentaires. Quand elles sont toutes satisfaites, les deux solutions sont optimales. La solution obtenue est bien entière, car à chaque itération, on s'assure que chaque modification maintient une solution primale entière.

Considérons maintenant une relaxation d'un problème **NP**-difficile. Cette relaxation n'a pas en général de solution optimale entière. Cela remet-il en cause l'approche par les conditions des écarts complémentaires ? La réponse est " non " : l'algorithme peut être modifié en relâchant convenablement ces conditions ! C'est la technique la plus courante pour concevoir des algorithmes d'approximation primal-dual ; mais ce n'est pas la seule.

15.1 Présentation générale du schéma primal-dual

Étudions le programme primal suivant, écrit sous forme standard.

$$\begin{aligned}
&\text{minimiser} && \sum_{j=1}^{n} c_j x_j && \\
&\text{sous les contraintes} && \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && i = 1, \ldots, m \\
& && x_j \geq 0, && j = 1, \ldots, n
\end{aligned}$$

où a_{ij}, b_i et c_j sont donnés. Son programme dual est :

$$\begin{array}{lll} \text{maximiser} & \displaystyle\sum_{i=1}^{m} b_i y_i & \\ \text{sous les contraintes} & \displaystyle\sum_{i=1}^{m} a_{ij} y_i \leq c_j, & j = 1, \ldots, n \\ & y_i \geq 0, & i = 1, \ldots, m \end{array}$$

La plupart des algorithmes primal-dual connus fonctionnent en assurant une partie des conditions, tout en relâchant convenablement les autres. Notre présentation couvre les deux situations car elle propose de relâcher les deux conditions. Par exemple, si les conditions primales sont vérifiées, il suffit de poser $\alpha = 1$; et si les conditions duales sont réalisées, $\beta = 1$.

Condition des écarts complémentaires primale relâchée
Soit $\alpha \geqslant 1$.
Pour tout $1 \leq j \leq n$: $\quad x_j = 0 \quad$ ou $\quad c_j/\alpha \leqslant \sum_{i=1}^{m} a_{ij} y_i \leqslant c_j$.

Condition des écarts complémentaires duale relâchée
Soit $\beta \geqslant 1$.
Pour tout $1 \leq i \leq m$: $\quad y_i = 0 \quad$ ou $\quad b_i \leqslant \sum_{j=1}^{n} a_{ij} x_j \leqslant \beta \cdot b_i$.

Proposition 15.1 *Si $\boldsymbol{x}$ et $\boldsymbol{y}$ sont deux solutions réalisables du primal et du dual (resp.) satisfaisant les conditions énoncées ci-dessus, alors*

$$\sum_{j=1}^{n} c_j x_j \leqslant \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i.$$

Preuve : La preuve utilise le mécanisme de paiement local (évoqué dans l'exercice 12.4) dont voici le principe. Supposons que la variable duale y_i possède une "somme d'argent" $\alpha\beta b_i y_i$. La somme possédée par la solution duale entière est donc égale au membre de droite de l'inégalité à démontrer. Nous allons utiliser la correspondance entre les variables duales et les contraintes primales, et réciproquement.

Les variables duales paient pour acheter la solution primale comme suit. y_i paie $\alpha y_i a_{ij} x_j$ pour la variable x_i. La somme totale déboursée par y_i est

$$\alpha y_i \sum_{j=1}^{n} a_{ij} x_j \leqslant \alpha\beta b_i y_i,$$

l'inégalité étant obtenue par les conditions des écarts complémentaires duales relâchées. Ainsi, l'argent déboursé par y_i est bien inférieur à celui qu'il détient en sa possession.

L'argent collecté par x_j vaut

$$\alpha x_j \sum_{i=1}^{m} a_{ij} y_i \geqslant c_j x_j,$$

l'inégalité provenant des conditions des écarts complémentaires primales relâchées. Par conséquent, les sommes d'argent perçues par les variables primales couvrent bien le coût de la solution primale. CQFD. □

L'algorithme commence avec une solutions primale non réalisable et une solution duale réalisable ; habituellement, avec les solutions triviales $\boldsymbol{x} = \mathbf{0}$ et $\boldsymbol{y} = \mathbf{0}$. Chaque étape "améliore la réalisibilité" de la solution primale d'une part, et la valeur de la solution duale d'autre part, de sorte qu'à la fin la solution primale soit réalisable et que les conditions des écarts complémentaires relâchées soient vérifiées pour des α et β convenables. La solution primale reste entière tout au long de l'algorithme, assurant ainsi que la solution finale soit entière. Enfin, le coût de la solution duale sert à minorer OPT, et donc à garantir que l'algorithme est une $\alpha\beta$-approximation, via la proposition 15.1.

Les améliorations des solutions primale et duale se font simultanément : la solution primale courante suggère comment améliorer la solution duale et réciproquement. Ces modifications sont "locales", dans l'esprit du mécanisme de paiement local de la proposition 15.1. Le paradigme sous-jacent de cette méthode peut se résumer ainsi : atteindre l'optimum global en construisant deux processus qui s'améliorent l'un l'autre localement.

15.2 Couverture par ensembles via le schéma primal-dual

Le schéma primal-dual va nous permettre d'obtenir une f-approximation pour la couverture par ensembles. Nous choisissons ici $\alpha = 1$ et $\beta = f$. Nous allons travailler sur la paire de programmes primal-dual (13.2) et (13.3). Les conditions des écarts complémentaires sont :

Conditions primales

$$\forall S \in \mathcal{S} : x_S \neq 0 \Rightarrow \sum_{e:\ e \in S} y_e = c(S).$$

Nous dirons qu'un ensemble S est *plein* si $\sum_{e:\ e \in S} y_e = c(S)$. Comme les variables primales seront entières à la fin de l'exécution de l'algorithme, nous pouvons reformuler ces conditions par : *ne sélectionner que des ensembles pleins*. De plus, comme la solution duale doit rester réalisable, aucun ensemble ne doit déborder.

Conditions duales

$$\forall e : y_e \neq 0 \Rightarrow \sum_{S:\ e \in S} x_S \leqslant f$$

Comme $\boldsymbol{x}$ sera à valeurs dans $\{0,1\}$, ces conditions peuvent se reformuler ainsi : *tout élément de valeur duale non-nulle est couvert au plus f fois.* Puisque tout élément appartient à au plus f ensembles, cette condition est vérifiée trivialement par tous les éléments.

Ces deux ensembles de conditions donnent naturellement l'algorithme suivant :

Algorithme 15.2 (Couverture par ensemble – f-approximation)

1. **Initialisation :** $\boldsymbol{x} \leftarrow \mathbf{0}$; $\boldsymbol{y} \leftarrow \mathbf{0}$
2. Répéter jusqu'à ce que tous les éléments soient couverts :

   Prendre un élément e non-couvert, et augmenter y_e jusqu'à ce qu'un des ensembles soit plein.

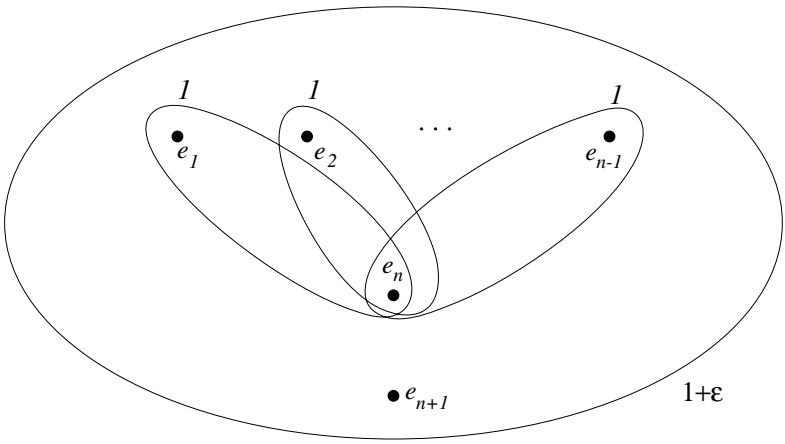
   Sélectionner tous les ensembles pleins dans la couverture, puis mettre à jour $\boldsymbol{x}$ en conséquence.

   Déclarer "couverts" tous les éléments de ces ensembles.
3. Renvoyer la couverture définie par $\boldsymbol{x}$.

Théorème 15.3 *L'algorithme 15.2 est une f-approximation.*

Preuve : Clairement, à la fin de l'algorithme, tous les éléments sont couverts et aucun ensemble ne déborde. Par conséquent, les solutions primale et duale sont toutes les deux réalisables. Puisqu'elles satisfont les conditions des écarts complémentaires relâchées pour $\alpha = 1$ et $\beta = f$, le facteur d'approximation est f, d'après la proposition 15.1. □

Exemple 15.4 Les ensembles suivants forment une instance critique pour cet algorithme :



L'ensemble $\mathcal{S}$ contient ici $n-1$ ensembles coûtant 1, $\{e_1, e_n\}, \ldots, \{e_{n-1}, e_n\}$, et un ensemble coûtant $1+\varepsilon$, $\{e_1, \ldots, e_{n+1}\}$, où $\varepsilon > 0$ est petit. Comme e_n appartient à n ensembles, nous avons $f = n$.

Supposons que l'algorithme augmente y_{e_n} à la première itération. Alors y_{e_n} est mis à 1, et tous les ensembles $\{e_i, e_n\}$, $1 \leqslant i \leqslant n-1$, sont pleins. Ces ensembles sont donc tous sélectionnés dans la couverture, qui couvre les éléments $e_1, \ldots, e_n$. Lors de la seconde itération, $y_{e_{n+1}}$ est mis à ε et l'ensemble $\{e_1, \ldots, e_{n+1}\}$ est plein. La couverture obtenue coûte donc $n+\varepsilon$, alors que l'optimum vaut $1+\varepsilon$. □

15.3 Exercices

15.1 Cet exercice est la suite de l'exercice 12.4. Démontrez que si $\boldsymbol{x}$ et $\boldsymbol{y}$ sont des solutions réalisables primale et duale (resp.) et satisfont les conditions énoncées à la section 15.2 avec $\alpha = 1$ et $\beta = f$, alors $\boldsymbol{y}$ achète $\boldsymbol{x}$ au prix f, c'est-à-dire

$$\sum_S c(S) x_S \leqslant f \cdot \sum_e y_e.$$

15.2 "Ôtez l'échafaudage" de programmation linéaire de l'algorithme 15.2 pour obtenir une f-approximation purement combinatoire pour la couverture par ensembles.
Indication : Reportez-vous à l'algorithme de l'exercice 2.11.

15.3 Soit k une constante fixée, étudions les instances du problème de la couverture par ensembles, où la fréquence maximale f est inférieure à k. L'algorithme 15.2 démontre que le saut intégral de la relaxation (13.2) est majoré par k pour ces instances. Exhibez des exemples qui démontrent que cette borne est asymptotiquement juste.
Indication : Prenez l'hypergraphe régulier G à n sommets, dont les hyper-arêtes sont tous les choix de k sommets parmi n. Puis, définissez l'instance ainsi : les sommets de l'instance sont les hyper-arêtes, et les ensembles sont les sommets de l'hypergraphe ; la relation d'inclusion est définie par la relation d'incidence dans l'hypergraphe.

15.4 Exhibez une famille d'instances critiques pour l'algorithme 15.2, pour laquelle f est bornée par une constante (dans l'exemple 15.4, f tend vers l'infini).

15.5 La relaxation suivante est exacte pour le problème du couplage de poids maximum (voir définition exercice 12.9) pour les graphes bipartis,

mais pas pour des graphes généraux. Proposez un algorithme utilisant le schéma primal-dual en relâchant convenablement les conditions des écarts complémentaires, pour démontrer que le saut intégral de cette relaxation est $\geqslant 1/2$. Quelle est la meilleure borne que vous puissiez donner sur le saut intégral ?

$$\begin{aligned} &\text{maximiser} && \sum_e w_e x_e && && (15.1)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e \leqslant 1, && v \in V\\ & && x_e \geqslant 0, && e \in E \end{aligned}$$

15.6 (Chudak, Goemans, Hochbaum, et Williamson [49]) Interprétez les algorithmes à base de stratification pour les problèmes de couverture par ensembles et coupe-cycles de sommets des chapitres 2 et 6 comme des algorithmes à base de schéma primal-dual. Comment les conditions des écarts complémentaires ont-elles été relâchées ?

15.4 Notes

Le premier algorithme primal-dual a été décrit par Kuhn [187] – pour le problème du couplage maximum dans un graphe biparti – qu'il a appelé la "méthode hongroise". Dantzig, Ford et Fulkerson [64] ont utilisé cette méthode pour construire de nouveaux algorithmes de résolution de programmes linéaires qu'ils ont baptisée "la méthode primal-dual". Bien que ce schéma n'ait pas rencontré un grand succès pour résoudre des programmes linéaires, il s'est répandu très rapidement en algorithmique combinatoire.

L'algorithme 15.2 est dû à Bar-Yehuda et Even [22]. Il n'était pas formulé comme un algorithme primal-dual mais c'est pourtant bien le premier algorithme d'approximation utilisant ce schéma. Les travaux de Agrawal, Klein et Ravi [1] et Goemans et Williamson [112] ont ravivé cette idée dans le contexte de l'approximation en l'améliorant par l'adjonction de l'idée puissante de l'augmentation synchrone des variables duales (voir chapitre 22). La relaxation des conditions des écarts complémentaires a été formalisée pour la première fois par Williamson, Goemans, Mihail et Vazirani [267]. Pour des compléments historiques, reportez-vous à Goemans et Williamson [114].

16 Satisfaction maximum

Le problème de la satisfaction maximum est un classique de l'approximation. Plus récemment, son étude a permis une amélioration considérable de notre connaissance de la difficulté d'obtenir une approximation (voir chapitre 29). Dans ce chapitre, nous proposons une 3/4-approximation fondée sur l'arrondi aléatoire, que nous dérandomiserons par la *méthode de l'espérance conditionnelle*.[1]

Problème 16.1 (Satisfaction maximum (MAX-SAT))[2] Étant donné une formule f sous forme normale conjonctive sur n variables booléennes $x_1, \ldots, x_n$ et munie de poids positifs w_c sur chaque clause c, trouver une instanciation des variables booléennes qui maximise le poids total des clauses satisfaites.

Notons $\mathcal{C}$ l'ensemble des clauses de f : $f = \bigwedge_{c \in \mathcal{C}} c$. Chaque clause est une disjonction de littéraux ; chaque littéral est une des variables booléennes ou sa négation. Notons taille(c) la *taille* de la clause c, c'est-à-dire le nombre de ses littéraux. Nous considérons que les tailles des clauses de f sont arbitraires. Les clauses sont supposées non triviales (par exemple, $c \neq x \vee y \vee \overline{x}$) et les littéraux non redondants à l'intérieur d'une même clause.

Pour tout entier positif k, nous appelons MAX-kSAT la restriction de MAX-SAT aux instances dont toutes les clauses sont de taille $\leqslant k$. MAX-SAT est **NP**-difficile ; en fait, même MAX-2SAT est **NP**-difficile (contrairement à 2SAT qui est dans **P**). Nous allons tout d'abord présenter deux algorithmes pour MAX-SAT : une 1/2-approximation et une $(1 - 1/e)$-approximation. La première est meilleure pour les clauses de grandes tailles, alors que la seconde est meilleure pour les petites clauses. Nous verrons ensuite comment combiner ces deux algorithmes pour obtenir la 3/4-approximation désirée.

Pour éviter une surcharge des notations, nous adoptons la même terminologie pour les trois algorithmes. Une même variable aléatoire W désignera le poids total des clauses satisfaites, et la variable aléatoire W_c sera la contribution au poids W de la clause c. Ainsi, $W = \sum_{c \in \mathcal{C}} W_c$ et

$$\mathbf{E}[W_c] = w_c \cdot \mathbf{Pr}[c \text{ est satisfaite}].$$

[1] *Method of conditional expectation*, en anglais.
[2] *Maximum satisfiability problem*, en anglais.

D'un point de vue formel strict, c'est un abus de notation car les lois de probabilité seront différentes pour les trois algorithmes.

16.1 Traitement des grandes clauses

Le premier algorithme est immédiat : donner indépendamment à chaque variable booléenne la valeur Vrai avec probabilité 1/2, et renvoyer l'instanciation τ obtenue. Pour $k \geqslant 1$, posons $\alpha_k = 1 - 2^{-k}$.

Lemme 16.2 *Si* taille$(c) = k$, *alors* $\mathbf{E}[W_c] = \alpha_k w_c$.

Preuve : La clause c n'est pas satisfaite par τ ssi tous ses littéraux valent Faux. La probabilité d'un tel événement vaut 2^{-k}. □

Pour $k \geqslant 1, \alpha_k \geqslant 1/2$. Donc d'après la linéarité de l'espérance,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geqslant \frac{1}{2} \sum_{c \in \mathcal{C}} w_c \geqslant \frac{1}{2} \text{OPT},$$

car le poids total des clauses de $\mathcal{C}$ majore trivialement OPT.

Plutôt que de transformer ce résultat en un énoncé avec forte probabilité, nous allons dérandomiser cette procédure. L'algorithme déterministe obtenu calculera une instanciation des variables telle que le poids des clauses satisfaites soit $\geqslant \mathbf{E}[W] \geqslant \text{OPT}/2$.

Remarquons que α_k croît avec k et que le facteur d'approximation garanti par cet algorithme est 3/4 si toutes les clauses ont au moins deux littéraux (l'algorithme suivant est conçu pour traiter plus efficacement les clauses unitaires).

16.2 Dérandomisation par la méthode de l'espérance conditionnelle

L'autoréductibilité de SAT (voir section A.5) joue un rôle crucial ici. Prenons un arbre d'autoréductibilité T de la formule f : les nœuds internes de niveau i correspondent aux instanciations des variables booléennes $x_1, \ldots, x_i$; et les feuilles représentent les instanciations complètes des n variables. Étiquetons chaque nœud de T par l'espérance conditionnelle du poids total des clauses satisfaites comme suit. Soit $a_1, \ldots, a_i$ une instanciation des variables $x_1, \ldots, x_i$. Le nœud correspondant à cette instanciation sera étiqueté par $\mathbf{E}[W | x_1 = a_1, \ldots, x_i = a_i]$. Si $i = n$, c'est une feuille, et son espérance conditionnelle est simplement le poids total des clauses satisfaites par cette instanciation.

Lemme 16.3 *On peut calculer en temps polynomial l'espérance conditionnelle en n'importe quel nœud de T.*

Preuve : Considérons un nœud $x_1 = a_1, \ldots, x_i = a_i$. Soit ϕ la formule booléenne sur les variables $x_{i+1}, \ldots, x_n$ obtenue à partir de ce nœud par autoréduction. L'espérance conditionnelle du poids des clauses de ϕ satisfaites par une instanciation aléatoire des variables $x_{i+1}, \ldots, x_n$ se calcule clairement en temps polynomial. Il suffit alors d'ajouter le poids des clauses de f déjà satisfaites par l'instanciation partielle $x_1 = a_1, \ldots, x_i = a_i$ pour obtenir le résultat souhaité. □

Théorème 16.4 *On peut calculer en temps polynomial un chemin de la racine de l'arbre à une feuille, tel que l'espérance conditionnelle en tout nœud de ce chemin soit $\geqslant \mathbf{E}[W]$.*

Preuve : L'espérance conditionnelle d'un nœud est la moyenne des espérances conditionnelles de ses enfants, c'est-à-dire

$$\begin{aligned}\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i] = \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{Vrai}]/2 + \\ \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{Faux}]/2.\end{aligned}$$

La raison en est, bien sûr, que x_{i+1} a la même probabilité de valoir Vrai ou Faux. Par conséquent, l'espérance conditionnelle pour le fils de plus grande espérance conditionnelle est supérieure à celle de son père, d'où l'existence d'un tel chemin. Enfin, par le lemme 16.3, ce chemin se calcule en temps polynomial. □

L'algorithme déterministe se déduit immédiatement du théorème 16.4. Il suffit de renvoyer l'instanciation de la feuille au bout du chemin calculé. Le poids total des clauses satisfaites sera bien $\geqslant \mathbf{E}[W]$.

Cette méthode de dérandomisation est très générale et peut être utilisée pour déterminiser des algorithmes randomisés plus complexes. Supposons que l'algorithme ne choisisse pas les valeurs des variables booléennes indépendamment les unes des autres (voir remarque 16.6 par exemple). Alors,

$$\begin{aligned}&\mathbf{E}[W|x_1 = a_1, ..., x_i = a_i] = \\ &\quad \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{V}] \cdot \mathbf{Pr}[x_{i+1} = \text{V}|x_1 = a_1, ..., x_i = a_i] \\ &+ \mathbf{E}[W|x_1 = a_1, ..., x_i = a_i, x_{i+1} = \text{F}] \cdot \mathbf{Pr}[x_{i+1} = \text{F}|x_1 = a_1, ..., x_i = a_i].\end{aligned}$$

La somme des deux probabilités conditionnelles vaut 1, car les deux événements sont complémentaires ($\leqslant 1$ suffirait). L'espérance conditionnelle du père est donc bien la combinaison convexe des espérances de ses enfants. Il en résulte que si on peut déterminer en temps polynomial lequel des deux est le plus grand, nous pouvons de nouveau dérandomiser l'algorithme. Néanmoins, le calcul des espérances conditionnelles n'est pas toujours

aisé. Remarquons que l'indépendance était fondamentale dans la preuve du lemme 16.3. C'est grâce à elle, qu'on pouvait considérer que l'instanciation des variables booléennes $x_{i+1}, \ldots, x_n$ est bien aléatoire uniforme, et donc calculer l'espérance du poids des clauses satisfaites de ϕ.

En général, un algorithme randomisé utilise un espace probabiliste plus complexe, éventuellement uniforme. Mais, comme l'espérance conditionnelle du père est toujours la combinaison convexe des espérances de ses fils (pondérés par leurs probabilités conditionnelles), il existe toujours un fils d'espérance supérieure (ou égale) à celle de son père.

16.3 Traitement des petites clauses par arrondi

Voici un programme linéaire entier encodant MAX-SAT. À chaque clause c de $\mathcal{C}$, nous associons l'ensemble S_c^+ (resp., S_c^-) des variables booléennes positives (resp., niées) de c. Une instanciation est codée par $\boldsymbol{y}$: $y_i = 1$ (resp., $y_i = 0$) signifie que x_i vaut Vrai (resp., Faux). La contrainte associée à chaque clause c assure que z_c ne vaut 1 uniquement si au moins l'un des littéraux de c est vrai, c'est-à-dire si la clause c est satisfaite par l'instanciation $\boldsymbol{y}$.

$$\begin{aligned} &\text{maximiser} && \sum_{c\in\mathcal{C}} w_c z_c && (16.1)\\ &\text{sous les contraintes} && \forall c \in \mathcal{C} : \sum_{i\in S_c^+} y_i + \sum_{i\in S_c^-} (1-y_i) \geqslant z_c \\ & && \forall c \in \mathcal{C} : \ z_c \in \{0,1\} \\ & && \forall i : \ y_i \in \{0,1\} \end{aligned}$$

Sa relaxation est :

$$\begin{aligned} &\text{maximiser} && \sum_{c\in\mathcal{C}} w_c z_c && (16.2)\\ &\text{sous les contraintes} && \forall c \in \mathcal{C} : \sum_{i\in S_c^+} y_i + \sum_{i\in S_c^-} (1-y_i) \geqslant z_c \\ & && \forall c \in \mathcal{C} : \ 1 \geqslant z_c \geqslant 0 \\ & && \forall i : \ 1 \geqslant y_i \geqslant 0 \end{aligned}$$

L'algorithme est immédiat : calculer une solution optimale $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ du programme linéaire (16.2) ; fixer chaque x_i à Vrai indépendamment avec probabilité y_i^*, pour $1 \leqslant i \leqslant n$; puis renvoyer l'instanciation τ obtenue.

Nous utilisons les variables aléatoires W et W_c définies à la section 16.1.

Pour $k \geqslant 1$, posons

$$\beta_k = 1 - \left(1 - \frac{1}{k}\right)^k.$$

Lemme 16.5 *Si* taille$(c) = k$, *alors*

$$\mathbf{E}[W_c] \geqslant \beta_k w_c z_c^*.$$

Preuve : Quitte à échanger x_i et $\overline{x}_i$ dans la formule, et quitte à modifier en conséquence le programme linéaire (16.2) (sans affecter z_c^*, ni W_c), nous pouvons supposer que toutes les littéraux sont positifs dans la clause c. Enfin, quitte à renommer les variables, nous supposons que $c = (x_1 \vee \cdots \vee x_k)$.

La clause c est satisfaite ssi $x_1, \ldots, x_k$ ne valent pas tous Faux. La probabilité d'un tel événement est

$$\begin{aligned} 1 - \prod_{i=1}^{k}(1 - y_i) &\geqslant 1 - \left(\frac{\sum_{i=1}^{k}(1 - y_i)}{k}\right)^k = 1 - \left(1 - \frac{\sum_{i=1}^{k} y_i}{k}\right)^k \\ &\geqslant 1 - \left(1 - \frac{z_c^*}{k}\right)^k. \end{aligned}$$

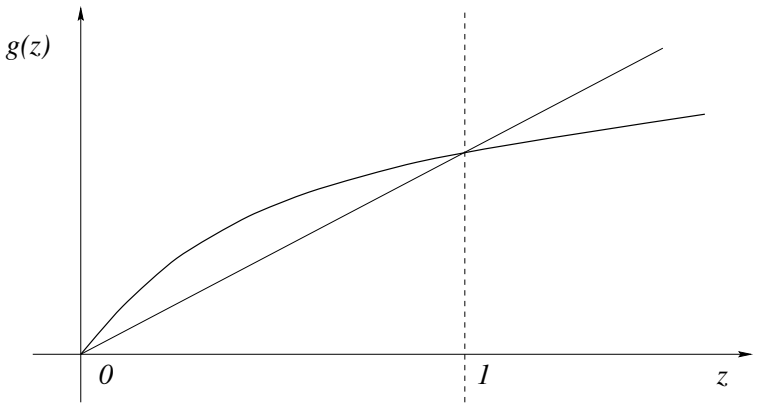
La première inégalité est donnée par l'inégalité arithmético-géométrique qui dit que pour tous réels positifs $a_1, \ldots, a_k$,

$$\frac{a_1 + \ldots + a_k}{k} \geqslant \sqrt[k]{a_1 \times \ldots \times a_k}.$$

La seconde inégalité utilise la contrainte $y_1 + \ldots + y_k \geqslant z_c$ du programme linéaire (16.2).

Soit g la fonction définie par :

$$g(z) = 1 - \left(1 - \frac{z}{k}\right)^k.$$



C'est une fonction concave telle que $g(0) = 0$ et $g(1) = \beta_k$. Par conséquent, pour tout $z \in [0,1]$, $g(z) \geqslant \beta_k z$. Ainsi, $\mathbf{Pr}[c \text{ est satisfaite}] \geqslant \beta_k z_c^*$. CQFD. □

Remarquons que β_k décroît avec k. Ainsi, si toutes les clauses sont de taille $\leqslant k$,

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geqslant \beta_k \sum_{c \in \mathcal{C}} w_c z_c^* = \beta_k \, \mathrm{OPT}_f \geqslant \beta_k \, \mathrm{OPT},$$

où OPT_f est la valeur d'une solution optimale du programme (16.2) (clairement, $\mathrm{OPT}_f \geqslant \mathrm{OPT}$). Cet algorithme peut également être dérandomisé par la méthode de l'espérance conditionnelle (voir exercice 16.3). Cet algorithme est donc une β_k-approximation pour les instances de MAX-SAT dont les clauses sont toutes de taille $\leqslant k$. Enfin, puisque

$$\forall k \in \mathbf{N} : \left(1 - \frac{1}{k}\right)^k < \frac{1}{e},$$

c'est une $(1 - 1/e)$-approximation pour MAX-SAT (pour toute instance).

16.4 Une 3/4-approximation

Voici comment combiner les deux algorithmes précédents. Soit b le résultat d'un tirage à pile-ou-face non biaisé. Si $b = 0$, alors on exécute le premier algorithme randomisé, et si $b = 1$, le second.

Remarque 16.6 Bien que nous fixions chaque x_i à Vrai avec probabilité $\frac{1}{4} + \frac{1}{2} y_i^*$, les valeurs des x_i ne sont *pas* choisies indépendamment ici !

Soit $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ une solution optimale du programme (16.2) pour une instance donnée.

Lemme 16.7 $\quad \mathbf{E}[W_c] \geqslant \dfrac{3}{4} w_c z_c^*$.

Preuve : Posons $k = \text{taille}(c)$. D'après le lemme 16.2,

$$\mathbf{E}[W_c \mid b = 0] = \alpha_k w_c \geqslant \alpha_k w_c z_c^*,$$

car $z_c^* \leqslant 1$. D'après le lemme 16.5,

$$\mathbf{E}[W_c \mid b = 1] \geqslant \beta_k w_c z_c^*.$$

En sommant ces inégalités, nous obtenons

$$\mathbf{E}[W_c] = \frac{1}{2}(\mathbf{E}[W_c \mid b = 0] + \mathbf{E}[W_c \mid b = 1]) \geqslant w_c z_c^* \frac{(\alpha_k + \beta_k)}{2}.$$

Or, $\alpha_1+\beta_1=\alpha_2+\beta_2=3/2$, et pour $k\geqslant 3$, $\alpha_k+\beta_k\geqslant 7/8+(1-1/e)\geqslant 3/2$. CQFD. □

Enfin, par la linéarité de l'espérance,

$$\mathbf{E}[W]=\sum_{c\in\mathcal{C}}\mathbf{E}[W_c]\geqslant\frac{3}{4}\sum_{c\in\mathcal{C}}w_c z_c^*=\frac{3}{4}\,\mathrm{OPT}_f\geqslant\frac{3}{4}\,\mathrm{OPT},\tag{16.3}$$

où OPT_f est la valeur d'une solution optimale du programme (16.2). Voici l'algorithme déterministe final.

Algorithme 16.8 (MAX-SAT – 3/4-approximation)

1. Soit τ_1, l'instanciation calculée par la 1/2-approximation dérandomisée.
2. Soit τ_2, l'instanciation calculée par la $(1-1/e)$-approximation dérandomisée.
3. Renvoyer la meilleure des deux instanciations.

Théorème 16.9 *L'algorithme 16.8 est une* 3/4*-approximation déterministe pour MAX-SAT.*

Preuve : D'après le lemme 16.7, la demi-somme des poids des clauses satisfaites par τ_1 et τ_2 est $\geqslant\frac{3}{4}\,\mathrm{OPT}$. La meilleure de ces deux instanciations fait donc au moins aussi bien. □

D'après (16.3), $\mathbf{E}[W]\geqslant\frac{3}{4}\,\mathrm{OPT}_f$. Le poids de la solution entière donnée par l'algorithme 16.8 est supérieur à $\mathbf{E}[W]$. Ainsi, le saut intégral de la relaxation (16.2) est $\geqslant 3/4$. Nous allons voir que cette borne est asymptotiquement juste.

Exemple 16.10 Prenons la formule $f=(x_1\vee x_2)\wedge(\overline{x}_1\vee x_2)\wedge(x_1\vee\overline{x}_2)\wedge(\overline{x}_1\vee\overline{x}_2)$, où chaque clause a un poids unitaire. Il est facile de voir que $y_i=1/2$ et $z_c=1$, pour tout i et c, est toujours une solution optimale du programme (16.2) pour toute instance dont les clauses sont de taille 2. Ainsi $\mathrm{OPT}_f=4$. Cependant, $\mathrm{OPT}=3$. Pour cette instance du programme (16.2), le saut intégral vaut donc 3/4. □

Exemple 16.11 Voici une instance critique pour l'algorithme 16.8. Soit $f=(x\vee y)\wedge(x\vee\overline{y})\wedge(\overline{x}\vee z)$, où les poids des trois clauses sont respectivement 1, 1, et $2+\varepsilon$. D'après la remarque faite à l'exemple 16.10, pour cette instance, la $(1-1/e)$-approximation, tout comme la 1/2-approximation va fixer à Vrai chaque variable avec probabilité 1/2. Supposons que durant la dérandomisation, la variable x soit fixée en premier. Les espérances conditionnelles sont $\mathbf{E}[W\mid x=\text{Vrai}]=3+\varepsilon/2$ et $\mathbf{E}[W\mid x=\text{Faux}]=3+\varepsilon$. Ainsi, x sera fixé à Faux. Mais

cela conduit à un poids total de $3 + \varepsilon$, alors que fixer x à Vrai donne un poids de $4 + \varepsilon$. Nous pouvons clairement obtenir une famille infinie de telles instances en dupliquant à volonté ces trois clauses avec de nouvelles variables. □

16.5 Exercices

16.1 L'algorithme de la section 16.1 est une α_k-approximation si toutes les clauses de l'instance sont de taille $\geqslant k$. Exhibez une instance critique à α_k de l'optimum pour cet algorithme.

16.2 Démontrez que l'algorithme suivant est une 1/2-approximation pour MAX-SAT. Soit τ une instanciation arbitraire des variables et τ' l'instanciation inverse, c'est-à-dire une variable vaut Vrai dans τ ssi elle vaut Faux dans τ' ; calculer le poids des clauses satisfaites par τ et τ', et renvoyer la meilleure des instanciations.

16.3 Appliquez la méthode de l'espérance conditionnelle pour dérandomiser la $(1 - 1/e)$-approximation pour MAX-SAT.

16.4 La 3/4-approximation ne choisit pas les valeurs des variables aléatoires indépendamment. Cependant, comme nous l'avons remarqué section 16.2, nous pouvons tout de même dérandomiser l'algorithme par la méthode de l'espérance conditionnelle. Expliquez comment. Remarquez que l'algorithme obtenu est différent de l'algorithme 16.8.

16.5 (Goemans et Williamson [111]) Au lieu d'utiliser directement la solution $\boldsymbol{y}^*$ du programme (16.2) pour fixer x_i à Vrai avec probabilité y_i^*, proposez d'utiliser $g(y_i^*)$, pour une fonction g bien choisie. Cela peut-il conduire à une amélioration du facteur d'approximation $(1 - 1/e)$?

16.6 Étudions l'algorithme randomisé suivant pour le problème de la coupe maximum (défini exercice 2.1) : après l'étape d'initialisation de l'algorithme 2.13, chacun des sommets restants est placé dans A ou dans B avec probabilité égale. Démontrez que l'espérance de la taille de la coupe obtenue est $\geqslant \mathrm{OPT}/2$. Démontrez que l'algorithme déterministe obtenu en dérandomisant cet algorithme par la méthode de l'espérance conditionnelle est précisément l'algorithme 2.13.

16.7 Étudions la généralisation suivante du problème de la coupe maximum.

Problème 16.12 (Systèmes d'équations linéaires sur GF[2])[3] Étant donné m équations à n variables sur GF[2], trouvez une instanciation des variables qui maximise le nombre d'équations linéaires satisfaites.

1. Démontrez que pour $m \leqslant n$, ce problème se résout en temps polynomial.
2. En général, ce problème est **NP**-difficile. Proposez une 1/2-approximation randomisée, que vous dérandomiserez par la méthode de l'espérance conditionnelle.

16.8 Étudions l'algorithme randomisé trivial pour MAX k-CUT (problème 2.14 de l'exercice 2.3) qui place aléatoirement chaque sommet dans les ensembles $S_1, \ldots, S_k$ avec probabilité égale. Démontrez que le nombre d'arêtes entre les ensembles obtenus est $\geqslant (1 - \frac{1}{k})\,\mathrm{OPT}$. Démontrez qu'en dérandomisant cet algorithme par la méthode de l'espérance conditionnelle, nous obtenons l'algorithme glouton de l'exercice 2.3.

16.9 Reprenez l'exercice 16.8 avec le problème de la coupe orientée maximum (problème 2.15 de l'exercice 2.4), c'est-à-dire proposez une 1/4-approximation randomisée pour ce problème, et démontrez que sa dérandomisation est précisément un algorithme glouton.

16.6 Notes

Johnson [158] est l'auteur de la 1/2-approximation, qui fut également la première approximation pour MAX-SAT. La première 3/4-approximation est due à Yannakakis [271]. L'algorithme (plus simple) que nous avons présenté est dû à Goemans et Williamson [111]. La méthode de l'espérance conditionnelle a été introduite implicitement par Erdös et Selfridge [81]. Puis, Spencer [252] a démontré son utilité pour l'obtention d'algorithmes polynomiaux (voir Raghavan [234], et Alon et Spencer [8] pour des raffinements de cette technique).

[3] *Linear equations over GF[2]*, en anglais.

17 Ordonnancement hétérogène

La technique d'arrondi des solutions fractionnaires d'un programme linéaire a donné de nombreux algorithmes d'approximation pour des problèmes d'ordonnancement **NP**-difficiles (voir section 17.6). Nous illustrons ce fait en proposant une 2-approximation pour le problème de l'ordonnancement hétérogène. Cet algorithme combine les techniques d'élagage paramétré, introduites au chapitre 5, et d'arrondi.

Problème 17.1 (Ordonnancement hétérogène)[1] Étant donné un ensemble de tâches J, un ensemble de machines M, et les temps d'exécution $p_{ij} \in \mathbf{N}$ de chaque tâche $j \in J$ sur chaque machine $i \in M$, ordonnancer les tâches sur les machines de façon à minimiser le *temps d'exécution total*,[2] c'est-à-dire la date de la fin d'exécution de la dernière tâche exécutée par les machines. Nous noterons n le nombre de tâches et m le nombre de machines.

Le mot « hétérogène » signifie que les temps de calcul d'une tâche sur les différentes machines se sont pas corrélés. Lorsque la durée de chaque tâche j est identique sur toutes les machines, notons-la p_j, les machines sont dites *homogènes* ou *identiques*. Ce problème a été étudié au chapitre 10 sous le nom du problème du temps d'exécution total minimum. Nous avons proposé un PTAS pour ce problème. Une variante de ce problème admet également un PTAS, il s'agit de l'ordonnancement sur machines *uniformes* (voir exercice 17.5). Dans cette variante, chaque machine i exécute les tâches à une vitesse s_i, c'est-à-dire le temps d'exécution de la tâche j sur la machine i vaut p_j/s_i.

17.1 Élagage paramétré et programmation linéaire

Notre problème est codé naturellement par le programme linéaire entier suivant. Dans ce programme, la variable x_{ij} indique si la tâche j est ordonnancée sur la machine i. Le but est de minimiser t, le temps d'exécution total. Le premier jeu de contraintes assure que toutes les tâches sont bien ordonnancées sur une machine, et le second que toutes les machines terminent leurs tâches avant t.

[1] *Scheduling on unrelated parallel machines*, en anglais.
[2] *Makespan*, en anglais.

$$\begin{array}{lll} \text{minimiser} & t & \\ \text{sous les contraintes} & \displaystyle\sum_{i\in M} x_{ij} = 1, & j \in J \\ & \displaystyle\sum_{j\in J} x_{ij}p_{ij} \leqslant t, & i \in M \\ & x_{ij} \in \{0,1\}, & i \in M,\ j \in J \end{array} \tag{17.1}$$

Le saut intégral de ce programme en nombres entiers n'est cependant pas borné.

Exemple 17.2 Considérons une instance avec une unique tâche, de même durée m sur les m machines. Le temps d'exécution minimum est clairement m. Cependant, la solution optimale de la relaxation linéaire, ordonnance une fraction $1/m$ de la tâche sur chaque machine, d'où un saut intégral de m. □

Cet exemple exploite une faille de la relaxation linéaire. Le programme entier fixe automatiquement x_{ij} à 0 si $p_{ij} > t$, alors que la relaxation linéaire est autorisée à donner des valeurs rationnelles pour obtenir une meilleure solution. Cette faille pourrait être comblée si l'on pouvait ajouter des contraintes du type :

$$\forall i \in M,\ j \in J :\ \text{si } p_{ij} > t,\ \text{alors}\ \ x_{ij} = 0.$$

Ces contraintes ne sont malheureusement pas linéaires.

La technique de l'élagage paramétré va nous permettre de contourner cette difficulté. Fixons un paramètre $T \in \mathbf{N}$, qui sera notre pari sur la valeur du temps d'exécution optimal. Ce paramètre va nous permettre d'éliminer toutes les paires tâche-machine telles que $p_{ij} > T$. Posons $S_T = \{(i,j) \mid p_{ij} \leqslant T\}$, et définissons à présent la famille de programmes linéaires $\mathrm{LP}(T)$, pour $T \in \mathbf{N}$. $\mathrm{LP}(T)$ utilise uniquement les x_{ij} pour $(i,j) \in S_T$, et s'interroge sur l'existence d'un ordonnancement fractionnaire de temps d'exécution $\leqslant T$ pour l'instance restreinte.

$$\begin{array}{lll} \text{minimiser} & t & \\ \text{sous les contraintes} & \displaystyle\sum_{i\,:\,(i,j)\in S_T} x_{ij} = 1, & j \in J \\ & \displaystyle\sum_{j\,:\,(i,j)\in S_T} x_{ij}p_{ij} \leqslant T, & i \in M \\ & x_{ij} \geqslant 0, & (i,j) \in S_T \end{array}$$

17.2 Propriétés des solutions extrémales

Une recherche dichotomique donne le plus petit T tel que LP(T) admette une solution réalisable. Notons-le T^*. T^* minore naturellement OPT. Ensuite, l'algorithme calcule, puis arrondit une solution extrémale de LP(T^*) pour construire un ordonnancement de temps d'exécution total $\leqslant 2T^*$. Les propriétés particulières des solutions extrémales de LP(T) nous seront très utiles.

Lemme 17.3 *Toute solution extrémale de* LP(T) *admet au plus* $n + m$ *variables non nulles.*

Preuve : Posons $r = |S_T|$, le nombre de variables réellement présentes dans LP(T). Rappelons qu'une solution réalisable LP(T) est extrémale ssi r contraintes linéairement indépendantes de LP(T) sont des égalités pour cette solution. Parmi ces r contraintes linéairement indépendantes, au moins $r - (n + m)$ doivent être choisies dans le troisième jeu de contraintes (de la forme $x_{ij} \geqslant 0$). Les variables correspondantes sont donc nulles. Ainsi toute solution extrémale admet au plus $n + m$ variables non nulles. □

Soit $\boldsymbol{x}$ une solution extrémale de LP(T). La tâche j est dite *entière dans* $\boldsymbol{x}$ si elle est ordonnancée intégralement sur une unique machine dans $\boldsymbol{x}$. Sinon, nous dirons que j est *fractionnaire*[3] *dans* $\boldsymbol{x}$.

Corollaire 17.4 *Au moins* $n - m$ *tâches sont entières dans toute solution extrémale de* LP(T).

Preuve : Soient $\boldsymbol{x}$ une solution extrémale de LP(T) et α et β les nombres de tâches entières et fractionnaires dans $\boldsymbol{x}$, respectivement. Les tâches fractionnaires sont ordonnancées sur deux machines au moins et sont donc associées à deux variables non nulles de $\boldsymbol{x}$ au moins. Par conséquent,

$$\alpha + \beta = n \text{ et } \alpha + 2\beta \leqslant n + m.$$

Ainsi, $\beta \leqslant m$ et $\alpha \geqslant n - m$. □

L'étape d'arrondi repose sur plusieurs propriétés combinatoires particulières des solutions extrémales de LP(T). Certaines de ces propriétés sont démontrées à la section 17.4. Nous associons à toute solution extrémale $\boldsymbol{x}$ de LP(T), un graphe biparti $G = (J, M, E)$, où $J \cup M$ est l'ensemble des sommets, et où $(j, i) \in E$ ssi $x_{ij} \neq 0$. Soient $F \subset J$ l'ensemble des tâches fractionnaires dans $\boldsymbol{x}$, et H le sous-graphe de G induit par les sommets $F \cup M$. Par définition, (i, j) est une arête de H ssi $0 < x_{ij} < 1$. Un couplage de H sera dit *parfait* si toute tâche $j \in F$ est couverte, c'est-à-dire associée à une machine par le couplage. L'étape d'arrondi repose sur l'existence d'un couplage parfait dans H (lemme 17.7).

[3] *Task j is integrally/fractionnally set in* $\boldsymbol{x}$, en anglais.

17.3 L'algorithme

La première étape de l'algorithme détermine un intervalle de recherche pour la valeur de T. Il s'agit de construire un ordonnancement glouton, où les tâches sont ordonnancées les unes après les autres sur la machine courante la moins chargée. Notons α le temps d'exécution total de cet ordonnancement. L'intervalle de recherche est alors fixé à $[\alpha/m, \alpha]$.

Algorithme 17.5 (Ordonnancement hétérogène)

1. Rechercher par dichotomie la plus petite valeur T^* de $T \in \mathbf{N}$ dans l'intervalle $[\alpha/m, \alpha]$, telle que $\mathsf{LP}(T)$ admette une solution réalisable.
2. Calculer une solution extrémale $\boldsymbol{x}$ de $\mathsf{LP}(T^*)$.
3. Ordonnancer toutes les tâches entières de $\boldsymbol{x}$ sur les machines données par $\boldsymbol{x}$.
4. Construire le graphe H et calculer un couplage parfait $\mathcal{M}$ de H (par exemple, utiliser la procédure décrite au lemme 17.7).
5. Ordonnancer les tâches fractionnaires sur les machines données par le couplage $\mathcal{M}$.

17.4 Propriétés particulières des solutions extrémales

Nous appelons *pseudo-arbre*, tout graphe connexe ayant au plus $|V|$ arêtes. Comme tout graphe connexe a plus de $|V|-1$ arêtes, un pseudo-arbre est, soit un arbre, soit un arbre plus une arête (et dans ce cas, admet un unique cycle). Un graphe est une *pseudo-forêt* si chacune de ses composantes connexes est un pseudo-arbre. Étudions maintenant les graphes G et H associés à toute solution extrémale $\boldsymbol{x}$ de $\mathrm{LP}(T)$, définie section 17.2.

Lemme 17.6 *Le graphe G est une pseudo-forêt.*

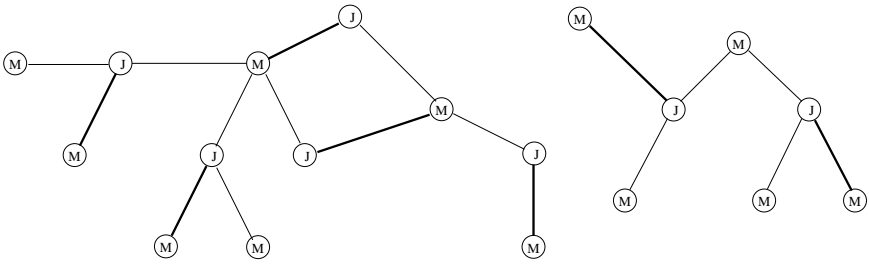
Preuve : Démontrons que le nombre d'arêtes de chaque composante connexe de G est borné par leur nombre de sommets, et donc que toute composante connexe est un pseudo-arbre.

Prenons une composante connexe G_c. Notons respectivement $\mathrm{LP}_c(T)$ et $\boldsymbol{x}_c$, le programme $\mathrm{LP}(T)$ et la solution $\boldsymbol{x}$, restreints aux tâches et machines de G_c. Soit $\boldsymbol{x}_{\overline{c}}$ le reste de $\boldsymbol{x}$. Le fait important est que $\boldsymbol{x}_c$ est une solution extrémale de $\mathrm{LP}_c(T)$. Raisonnons par l'absurde. Si $\boldsymbol{x}_c$ était une combinaison convexe de deux solutions réalisables distinctes de $\mathrm{LP}_c(T)$, chacune d'elles, complétée par $\boldsymbol{x}_{\overline{c}}$, serait une solution réalisable de $\mathrm{LP}(T)$. Ainsi, $\boldsymbol{x}$ serait combinaison convexe de deux solutions réalisables distinctes de $\mathrm{LP}(T)$, contradiction.

Le lemme 17.3 permet alors de conclure que G_c est bien un pseudo-arbre. □

Lemme 17.7 *Le graphe H admet un couplage parfait.*

Preuve : Les tâches entières dans $\boldsymbol{x}$ ont une unique arête dans G. En ôtant ces tâches et leurs arêtes de G, nous obtenons exactement le graphe H. Puisqu'on a ôté autant d'arêtes que de sommets de G, H est aussi une pseudo-forêt.



Dans H, les tâches sont de degré $\geqslant 2$. Ainsi, les feuilles de H sont nécessairement des machines. Couplons donc chaque machine feuille à sa tâche incidente, et retirons-les du graphe, et répétons ce processus tant que cela est possible (à chaque étape, toutes les feuilles sont des machines). À la fin de ce processus, il ne reste qu'un cycle pair (car H est biparti). En prenant une arête sur deux de ce cycle, nous obtenons un couplage parfait de H. □

Théorème 17.8 *L'algorithme 17.5 est une 2-approximation pour le problème de l'ordonnancement hétérogène.*

Preuve : Tout d'abord, l'algorithme est clairement polynomial. Ensuite, comme LP(OPT) est réalisable, $T^* \leqslant \mathrm{OPT}$. Le temps d'exécution total fractionnaire de la solution extrémale $\boldsymbol{x}$ de LP(T^*) est $\leqslant T^*$. Ainsi, le temps d'exécution de l'ordonnancement (entier) restreint aux tâches entières de $\boldsymbol{x}$ est $\leqslant T^*$. Pour chaque arête (i, j) de H, $p_{ij} \leqslant T^*$. Le couplage parfait calculé dans H ordonnance au plus une tâche supplémentaire sur chaque machine. Ainsi, le temps total d'exécution est $\leqslant 2 \cdot T^* \leqslant 2 \cdot \mathrm{OPT}$. □

Exemple 17.9 Voici une famille d'instances critiques. Soient $(m^2 - m + 1)$ tâches à ordonnancer sur m machines. La durée de la première tâche est m, et la durée des autres tâches est 1, indépendamment des machines. L'ordonnancement optimal place la première tâche sur une machine, et m des tâches restantes sur chacune des autres $m - 1$ machines. Le temps d'exécution total est m.

Il est facile de voir que LP(T) n'est pas réalisable pour $T < m$. Supposons que l'algorithme choisisse la solution extrémale de LP(m), qui ordonnance une fraction $1/m$ de la première tâche sur chaque machine et $m-1$ des tâches restantes sur chaque machine. L'étape d'arrondi conduit à un ordonnancement de temps d'exécution $2m-1$. □

17.5 Exercices

17.1 Proposez une autre preuve du lemme 17.7 utilisant le théorème de Hall. (Ce théorème affirme que tout graphe biparti $G = (U, V, E)$ admet un couplage couvrant tous les sommets de U ssi pour tout sous-ensemble $U' \subseteq U$, le cardinal du voisinage de U' est supérieur au cardinal de U' (le voisinage de U' est $\{v \in V \mid \exists u \in U',\ (u, v) \in E\}$).
Indication : Soit $F' \subset F$, et M' le voisinage de F'. Montrez que le graphe induit par $F' \cup M'$ admet au plus $|F'| + |M'|$ arêtes, puis que ce graphe a plus de $2|F'|$ arêtes, car le degré de tout sommet de F est $\geqslant 2$.

17.2 Démontrez que la solution de LP(m) proposée dans l'exemple 17.9 est bien extrémale.

17.3 Est-ce que le facteur d'approximation de l'algorithme 17.5 est meilleur que 2, lorsque les machines sont toutes identiques ?

17.4 Démontrez le raffinement suivant du lemme 17.6 : il existe une solution extrémale de LP(T), telle que son graphe biparti associé, G, soit une forêt.

17.5 (Hochbaum et Shmoys [137]) Proposez un PTAS pour le problème du temps d'exécution minimum sur machines uniformes. Dans ce problème, chaque machine i exécute les tâches à une vitesse s_i, c'est-à-dire exécute la tâche j en temps p_j/s_i.

17.6 Notes

Le résultat de ce chapitre est dû à Lenstra, Shmoys, et Tardos [192]. Se reporter à l'état de l'art de Hall [128] pour d'autres algorithmes d'ordonnancement à base d'arrondi d'une solution d'un programme linéaire.

18 Multicoupe et multiflot entier dans un arbre

La théorie des coupes dans les graphes joue un rôle central en algorithmique exacte, il en va de même en algorithmique d'approximation. Les quatre chapitres qui suivent présentent plusieurs résultats clés. Nous y retrouverons les méthodes développées aux chapitres 14 et 15.

Au chapitre 15, nous avions obtenu une 2-approximation pour le problème de la couverture pondérée par sommets, via le schéma primal-dual. Cet algorithme était particulièrement simple, car toute solution entière vérifiait systématiquement les conditions des écarts complémentaires. Dans ce chapitre, nous allons construire par schéma primal-dual un algorithme pour une généralisation de ce problème (voir exercice 18.1). Cette fois-ci, l'étape difficile sera de garantir les conditions des écarts complémentaires. Nous introduirons également une procédure d'élimination arrière, que l'on retrouve dans d'autres algorithmes primal-dual.

18.1 Les problèmes et leurs relaxations

Le problème suivant est une généralisation importante du problème de la coupe minimum de s à t. En fait, c'est également une généralisation du problème de la coupe multiséparatrice[1] (problème 4.1).

Problème 18.1 (Multicoupe minimum) Étant donné un graphe non orienté $G{=}(V, E)$, où chaque arête $e \in E$ a une capacité positive c_e, et un ensemble de paires de sommets $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, telles que $s_i \neq t_i$ pour tout i, une *multicoupe*[2] est un ensemble d'arêtes dont le retrait du graphe sépare les sommets de chaque paire. Le problème est de trouver une multicoupe de capacité minimum dans G.

Le problème de la coupe minimum de s à t est le cas particulier de la multicoupe où $k = 1$. Le problème de la coupe multiséparatrice minimum est également un cas particulier du problème 18.1, car séparer tous les terminaux $s_1, \ldots, s_\ell$ est équivalent à séparer toutes les paires (s_i, s_j), pour $1 \leqslant i < j \leqslant \ell$. Le problème de la multicoupe minimum est donc **NP**-difficile, même pour

[1] *Multiway cut*, en anglais.

[2] A *multicut*, en anglais.

$k = 3$, car la coupe multiséparatrice minimum est **NP**-difficile avec trois terminaux.

Le chapitre 20 présentera une $O(\log k)$-approximation pour la multicoupe minimum. Ce chapitre donne une 2-approximation pour le cas où G est un arbre. Lorsque G est un arbre, il existe un unique chemin entre s_i et t_i, une multicoupe doit donc sélectionner une arête de ce chemin pour séparer s_i et t_i. Bien que cette restriction du problème ait l'air désespérément triviale, il n'en est rien. L'exercice 18.1 démontre en particulier qu'elle est **NP**-difficile même si l'arbre est de hauteur 1 avec des coûts unitaires sur les arêtes.

Afin de pouvoir utiliser la théorie de la dualité, commençons par formuler notre problème sous forme d'un programme linéaire en nombres entiers, que nous relâcherons. Associons à chaque arête $e \in E$ une variable d_e à valeurs dans $\{0, 1\}$; $d_e = 1$ signifie que e est sélectionnée dans la multicoupe. Notons p_i l'unique chemin entre s_i et t_i dans l'arbre.

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{e \in E} c_e d_e & \\ \text{sous les contraintes} & \displaystyle\sum_{e \in p_i} d_e \geqslant 1, & i \in \{1, \ldots, k\} \\ & d_e \in \{0, 1\}, & e \in E \end{array}$$

Sa relaxation est obtenue en remplaçant la contrainte $d_e \in \{0, 1\}$ par $d_e \geqslant 0$. De même que dans la relaxation du programme (13.2), il est inutile d'ajouter la contrainte $d_e \leqslant 1$ explicitement.

$$\begin{array}{lll} \text{minimiser} & \displaystyle\sum_{e \in E} c_e d_e & \\ \text{sous les contraintes} & \displaystyle\sum_{e \in p_i} d_e \geqslant 1, & i \in \{1, \ldots, k\} \\ & d_e \geqslant 0, & e \in E \end{array} \tag{18.1}$$

d_e est maintenant la proportion de e sélectionnée dans la multicoupe. Une solution du programme relâché est appelée une *multicoupe fractionnaire* : la somme des proportions des arêtes sélectionnées le long de chaque chemin p_i, est supérieure à 1. En général, une multicoupe fractionnaire minimum est strictement moins chère qu'une multicoupe entière minimum (voir exemple 18.2).

Le programme dual s'interprète comme un problème de *multiflot*[3] dans G, où chaque flot va de s_i à t_i. La variable duale f_i correspond au flot particulier circulant le long de l'unique chemin de s_i à t_i.

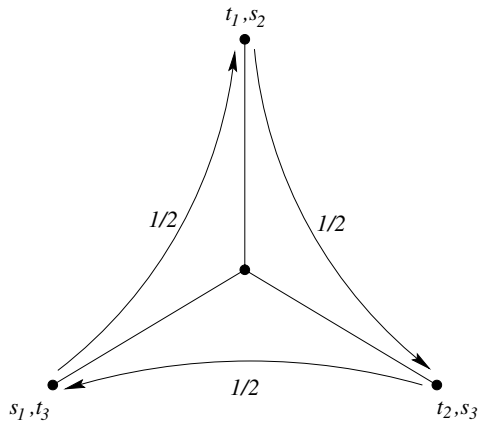
[3] The *multicommodity flow problem*, en anglais.

$$\begin{aligned} &\text{maximiser} && \sum_{i=1}^{k} f_i && && (18.2)\\ &\text{sous les contraintes} && \sum_{i\,:\,e\in p_i} f_i \leqslant c_e, && e \in E \\ & && f_i \geqslant 0, && i \in \{1,\ldots,k\} \end{aligned}$$

Les différents flots sont concurrents. Le but est de maximiser la somme des différents flots, sachant que la somme des flots traversant une arête (dans un sens ou dans l'autre) doit être inférieure (ou égale) à la capacité de l'arête.

D'après le théorème de dualité faible, tout multiflot réalisable donne une minoration de la capacité d'une multicoupe fractionnaire minimum et donc de la multicoupe entière optimale. Et d'après le théorème de dualité, la capacité minimum d'une multicoupe fractionnaire est la valeur du multiflot maximum.

Exemple 18.2 Étudions l'instance suivante, où les arêtes du graphe ont des capacités unitaires, et où on cherche à séparer les trois paires de sommets :



Les flèches indiquent comment faire passer 3/2 unités de flots en envoyant 1/2 unité de chaque flot. Sélectionner 1/2 de chaque arête donne une multicoupe fractionnaire de capacité 3/2 également. Ces solutions sont donc optimales pour les programmes dual et primal, respectivement. Cependant, toute multicoupe entière sélectionne nécessairement au moins deux arêtes pour séparer les trois paires, et a donc une capacité supérieure à 2. □

Définissons le problème suivant.

Problème 18.3 (Multiflot entier)[4] L'entrée consiste en un graphe G et des paires source-puits, définies comme dans le problème de la multicoupe minimum ; les capacités des arêtes sont cependant toutes entières. Un flot

[4] The *integer multicommodity flow problem*, en anglais.

différent est associé à chaque paire (s_i, t_i). Le but est de maximiser la somme des flots en circulation, étant donné les contraintes de capacité des arêtes et sachant que les flots doivent tous être entiers.

Nous étudions ici la restriction de ce problème au cas où G est un arbre. Nous obtenons un programme en nombres entiers pour ce problème en modifiant le programme (18.2) pour imposer des valeurs entières aux variables. La valeur objectif de ce programme en nombres entiers est majorée par celle du programme linéaire (18.2). En général, un flot fractionnaire optimal peut être strictement meilleur. Dans l'exemple 18.2, le multiflot entier maximum vaut 1 (on sature deux des trois arêtes en envoyant une unité de n'importe quel flot). Ce problème est **NP**-difficile, même restreint aux arbres de hauteur 3 (les capacités doivent cependant être arbitraires).

18.2 Algorithme primal-dual

Nous allons utiliser la méthode primal-dual pour obtenir un algorithme qui trouve simultanément une multicoupe et un multiflot entier, tous deux à un facteur 2 de l'optimum, pour peu que le graphe soit un arbre. Cet algorithme sera respectivement une 2- et une 1/2-approximation pour ces deux problèmes.

Prenons pour programme primal, le programme de la multicoupe. Nous dirons qu'une arête e est *saturée* si le flot total qui la traverse est égal à sa capacité. Nous allons garantir les conditions primales des écarts complémentaires, c'est-à-dire $\alpha = 1$, et relâcher les conditions duales en prenant $\beta = 2$ (où α et β sont les paramètres introduits pour cette méthode au chapitre 15).

Conditions primales : Pour tout $e \in E$, $d_e \neq 0 \Rightarrow \sum_{i\,:\,e \in p_i} f_i = c_e$.
De manière équivalente, *toutes les arêtes choisies dans la multicoupe sont saturées.*

Conditions duales relâchées : Pour tout $i \in \{1, \ldots, k\}$, $f_i \neq 0 \Rightarrow \sum_{e \in p_i} d_e \leq 2$.
De manière équivalente, *au plus deux arêtes sont choisies sur tout chemin transportant un flot non nul* — nous devons sélectionner au moins une arête sur chaque chemin de s_i à t_i pour obtenir une multicoupe réalisable.

Enracinons l'arbre G en un sommet arbitraire. La *profondeur* d'un sommet v est sa distance à la racine ; la profondeur de la racine est zéro. Le *plus petit ancêtre commun* $\mathrm{ppac}(u, v)$ de deux sommets u et v, est le sommet le moins profond sur l'unique chemin entre u et v. Soient deux arêtes e_1 et e_2 sur le chemin d'un sommet à la racine, nous dirons que e_1 est *plus profonde* que e_2, si e_1 est avant e_2 sur ce chemin.

L'algorithme démarre avec une multicoupe vide et un flot nul. Chaque étape améliore la réalisabilité de la solution primale et la valeur de la solution duale. Durant chaque itération, l'algorithme sélectionne le nœud v le plus profond qu'il n'a pas encore traité, et ajoute gloutonnement un flot

entier entre les paires de sommets qui admettent v pour plus petit ancêtre commun. Quand plus aucun flot ne pourra être routé entre ces paires, toutes les arêtes saturées par cette itération sont placées dans une liste D, dans un ordre arbitraire. Lorsque tous les sommets ont été traités, D est une multicoupe ; cependant, D contient certaines redondances. L'algorithme supprime ces redondances par une phase d'*élimination en arrière*:[5] les arêtes de D sont parcourues dans l'ordre inverse de leur insertion, et si l'élimination de l'arête e de D donne toujours une multicoupe, l'arête e est ôtée de D.

Algorithme 18.4 (Multicoupe et Multiflot entier dans un arbre)

1. **Initialisation :** $f \leftarrow \mathbf{0}$; $D \leftarrow \varnothing$.
2. **Routage du flot :** Pour chaque sommet v, pris par profondeur décroissante, faire :

   Pour tout (s_i, t_i) tel que $\mathrm{ppac}(s_i, t_i) = v$, ajouter gloutonnement à f un flot entier maximum entre s_i et t_i.

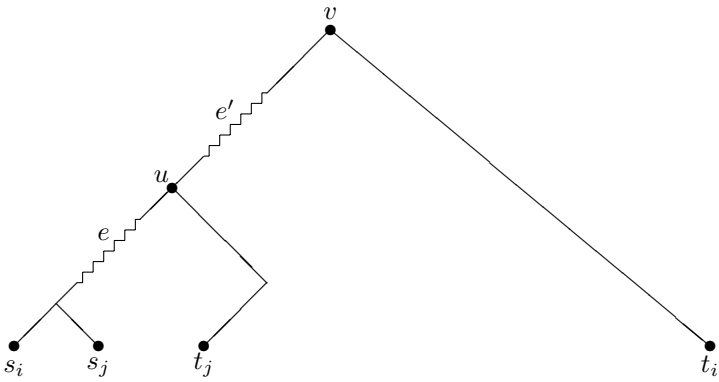
   Ajouter dans D toutes les arêtes saturées dans cette itération, dans un ordre arbitraire.
3. Soit $(e_1, e_2, \ldots, e_\ell)$ la liste des arêtes de D, ordonnées par dates d'insertion croissante.
4. **Élimination en arrière :** Pour j décroissant de ℓ à 1 faire :

   Si $D - \{e_j\}$ est une multicoupe de G, alors $D \leftarrow D - \{e_j\}$.
5. Renvoyer le multiflot f et la multicoupe D.

Lemme 18.5 *Soit (s_i, t_i) une paire dont le flot soit non nul, et posons $v = \mathrm{ppac}(s_i, t_i)$. Alors, au plus une arête a été sélectionnée dans la multicoupe, sur chacun des chemins de s_i à v et de t_i à v.*

Preuve : La preuve est identique pour chaque chemin. Supposons que deux arêtes e et e' soient sélectionnées sur le chemin s_i–v, et que e soit plus profonde que e'. Clairement, e' appartient à D tout le long de l'étape d'élimination en arrière. Étudions l'instant où e est testée dans l'étape d'élimination en arrière. Puisque e n'est pas éliminée, il existe une paire (s_j, t_j), telle que e soit l'unique arête de D sur le chemin s_j–t_j. Notons u le plus petit ancêtre commun de s_j et t_j. Puisque e' n'est pas sur le chemin s_j–t_j, u est plus profond que e', et donc plus profond que v. À la fin du traitement de u, D devait donc contenir une arête e'' sur le chemin s_j–t_j.

[5] *Reverse delete step*, en anglais.



Puisqu'un flot non nul a été routé entre s_i et t_i, e a dû être ajoutée pendant ou après le traitement de v. Comme v est un ancêtre de u, e a été ajoutée après e''. Ainsi e'' appartient à D quand e est testée, ce qui contredit le fait que e soit la seule arête de D sur le chemin s_j–t_j. □

Théorème 18.6 *Lorsque le graphe est un arbre, l'algorithme 18.4 est une* 2*-approximation pour le problème de la multicoupe minimum et une* 1/2*-approximation pour le problème du multiflot entier maximum.*

Preuve : Le flot construit par l'étape 2 est maximal. Puisque D contient toutes les arêtes saturées à la fin de cette étape, D est une multicoupe. Comme l'étape d'élimination en arrière supprime uniquement des arêtes redondantes, D est toujours une multicoupe à la fin de l'algorithme. Les solutions renvoyées sont donc bien respectivement un flot et une multicoupe réalisables.

Comme toutes les arêtes de la multicoupe sont saturées, les conditions primales sont vérifiées. D'après le lemme 18.5, dans tout chemin transportant un flot non nul, au plus deux arêtes sont sélectionnées dans la multicoupe. Par conséquent, les conditions duales relâchées sont satisfaites également. Ainsi, d'après la proposition 15.1, la capacité de la multicoupe renvoyée est majorée par le double de la valeur du flot. Enfin, la valeur de tout flot réalisable minore la capacité d'une multicoupe optimale, et la capacité de toute multicoupe réalisable majore la valeur de tout multiflot entier. CQFD. □

Nous tirons du théorème 18.6, l'inégalité min-max suivante :

Corollaire 18.7 *Pour un arbre muni de capacités entières sur les arêtes,*

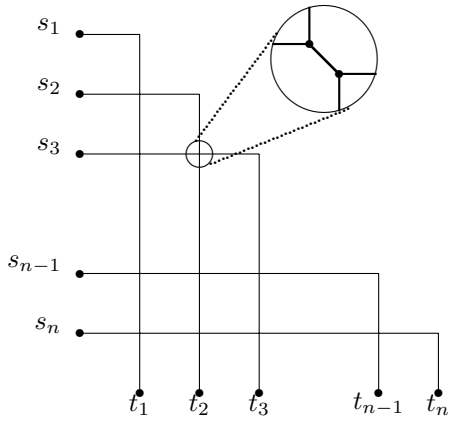
$$\max_{\text{flot entier } F} |F| \quad \leqslant \min_{\text{multicoupe } C} c(C) \quad \leqslant 2 \max_{\text{flot entier } F} |F|,$$

où $|F|$ *et* $c(C)$ *sont respectivement la valeur du flot* F*, et la capacité de la multicoupe* C*.*

Nous présenterons au chapitre 20, une $O(\log k)$-approximation pour le problème de la multicoupe minimum dans un graphe arbitraire ; le minorant

utilisé sera, une fois encore, une multicoupe fractionnaire optimale. En revanche, aucune approximation non triviale n'est connue actuellement pour le problème du multiflot entier pour des graphes plus généraux que des arbres. Même pour les graphes planaires (exemple 18.8), le saut intégral d'une relaxation similaire à (18.2) est minoré par $n/2$, où n est le nombre de paires source-puits.

Exemple 18.8 Étudions le graphe planaire suivant, avec n paires source-puits $(s_1, t_1), \ldots, (s_n, t_n)$ et où toutes les arêtes ont une capacité unitaire. Toute paire de chemins entre la i-ième et la j-ième paire source-puits passe par une même arête de capacité 1. L'agrandissement montre la disposition des arêtes aux intersections. L'envoi d'une unité de n'importe quel flot bloque donc tous les autres flots, alors que l'envoi simultané d'une demi-unité de chaque flot est possible.



□

18.3 Exercices

18.1 (Garg, Vazirani, et Yannakakis [105]) Exhibez une réduction isofacteur pour les paires suivantes de problèmes :

(a) la couverture par sommets de taille minimum, et la multicoupe minimum dans un arbre de hauteur 1 avec des capacités unitaires sur les arêtes ;

(b) la couverture pondérée par sommets (avec des poids arbitraires), et la multicoupe minimum dans un arbre de hauteur 1 avec des capacités arbitraires sur les arêtes.

Indication : Étant donné une instance G de la couverture par sommets, construire un arbre de hauteur 1, qui a une feuille pour chaque sommet de G et une paire source-puits pour chaque arête de G.

18.2 Le problème suivant est une généralisation très étudiée du problème du couplage maximum, qui se résout en temps polynomial. Étant donné un graphe non orienté $G = (V, E)$ et une fonction $b : V \to \mathbf{N}$, un b-couplage est un multi-ensemble d'arêtes $E' \subseteq E$ de multiplicité $m : E' \to \mathbf{N}$, tel que le nombre d'arêtes incidentes à tout sommet $v \in V$ est inférieur à $b(v)$, multiplicités comprises. La taille d'un b-couplage est la somme des multiplicités des arêtes de E'. Le *problème du b-couplage maximum*[6] est de trouver un b-couplage de taille maximum. Démontrez que les paires suivantes de problèmes sont équivalentes par réductions polynomiales :

(a) le problème du multiflot entier dans un arbre de hauteur 1 avec des arêtes de capacités unitaires, et le problème du couplage maximum ;

(b) le problème du multiflot entier maximum sur un arbre de hauteur 1 avec des arêtes de capacités arbitraires, et le problème du b-couplage maximum.

18.3 (Garg, Vazirani, et Yannakakis [105]) Proposez un algorithme polynomial pour calculer un multiflot entier dans un arbre de hauteur arbitraire avec des arêtes de capacités unitaires.
Indication : Procédez par programmation dynamique, et utilisez une procédure de couplage maximum.

18.4 Démontrez que si l'étape 2 de l'algorithme 18.4 est modifiée pour n'inclure qu'une unique arête saturée à chaque itération, alors l'ensemble D obtenu n'est pas nécessairement une multicoupe.

18.5 Démontrez que si l'étape 4 de l'algorithme 18.4 n'est pas exécutée, ou bien est modifiée pour éliminer en avant, alors le facteur d'approximation n'est plus borné.

18.6 Modifiez l'étape 4 de l'algorithme 18.4 en : trier les arêtes de D par capacité décroissante et éliminer les arêtes redondantes dans cet ordre. Quel facteur d'approximation pouvez-vous garantir pour cet algorithme ?

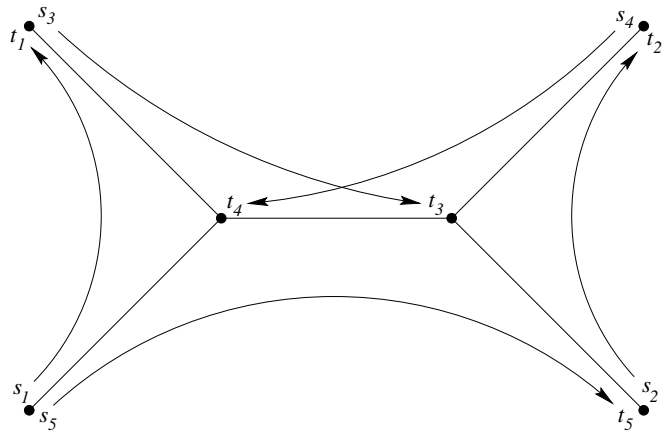
18.7 Exhibez des instances critiques de l'algorithme 18.4 pour les deux problèmes de la multicoupe et du multiflot entier.

18.8 Démontrez que si e et e' appartiennent à D à l'étape 3 de l'algorithme 18.4, et que e est plus profonde que e', alors e est insérée dans D avant ou à la même itération que e'.

18.9 Trouvez les meilleures multicoupes entière et fractionnaire, et le meilleur multiflot dans le graphe suivant. Toutes les capacités sont unitaires, et les paires source-puits sont $(s_1, t_1), \ldots, (s_5, t_5)$.

[6] The *maximum b-matching problem*, en anglais.

Remarquez que la multicoupe fractionnaire optimale n'est pas demi-entière, alors que la relaxation pour le problème de la coupe multiséparatrice admet toujours une solution optimale demi-entière (voir chapitre 19).



18.4 Notes

L'algorithme 18.4 est dû à Garg, Vazirani et Yannakakis [105]. Se reporter à Guruswami, Khanna, Rajaraman, Shepherd et Yannakakis [126] pour des résultats récents sur le problème du multiflot entier.

19 Coupe multiséparatrice

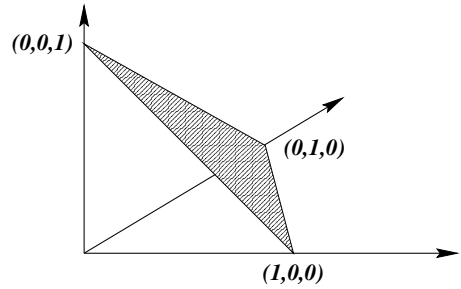
Nous avons obtenu au chapitre 4, une $(2-2/k)$-approximation combinatoire simple pour le problème de la coupe multiséparatrice[1] (problème 4.1). Dans ce chapitre, nous proposons une 3/2-approximation fondée sur l'arrondi randomisé.

Nous avons vu au chapitre 14, la propriété remarquable de demi-intégralité, vérifiée par certaines relaxations de programmes linéaires associés à des problèmes **NP**-difficiles. Les problèmes de la coupe multiséparatrice et de sa généralisation, la coupe de nœuds multiséparatrice[2] ont cette propriété. Nous le démontrons à la section 19.3. C'est d'ailleurs l'unique moyen connu à ce jour pour obtenir une approximation à un facteur constant pour le second problème.

19.1 Une relaxation intéressante

Le saut intégral de la relaxation usuelle pour la coupe multiséparatrice est $2-2/k$ (voir exercice 19.2). Pour obtenir un meilleur facteur d'approximation, nous allons utiliser une nouvelle et astucieuse relaxation.

Notons Δ_k le simplexe de dimension $k-1$. Il s'agit du polyèdre convexe de dimension $k-1$ dans $\mathbf{R}^k$ défini par $\{\boldsymbol{x} \in \mathbf{R}^k \,:\, \boldsymbol{x} \geqslant 0 \text{ et } \sum_i x_i = 1\}$, où x_i est la i-ième coordonnée du point $\boldsymbol{x}$. Le simplexe Δ_3 est illustré ci-dessous.



La nouvelle relaxation associe à chaque sommet de G un point de Δ_k. Chacun des k terminaux est associé à un sommet distinct du simplexe,

[1] *Multiway cut*, en anglais.
[2] *Node multiway cut*, en anglais.

c'est-à-dire à un vecteur de la base canonique (e_i) de $\mathbf{R}^k$. Notons $x_v \in \Delta_k$ le point associé au sommet v du graphe. La longueur $d(uv)$ de l'arête $uv \in E$ est définie comme la moitié de la distance ℓ_1 entre x_u et x_v. Voici la relaxation :

$$\begin{aligned} &\text{minimiser} && \sum_{uv \in E} c(uv) d(uv) && (19.1)\\ &\text{sous les contraintes} && d(uv) = \frac{1}{2} \sum_{i=1}^{k} |x_u^i - x_v^i|, && uv \in E \\ & && x_v \in \Delta_k, && v \in V \\ & && x_{s_i} = e_i, && s_i \in S \end{aligned}$$

Le lemme 19.1 démontre que cette relaxation est bien linéaire. Une solution entière de cette relaxation associe à chaque sommet de G un sommet du simplexe. Chaque arête uv a alors pour longueur 1 ou 0, suivant que u et v sont associés ou non au même sommet du simplexe. La coupe multiséparatrice se compose des arêtes de longueur 1. Le coût de cette coupe est la valeur objectif de cette solution entière. Ainsi, toute solution entière optimale correspond bien à une coupe multiséparatrice optimale.

Lemme 19.1 *La relaxation (19.1) s'écrit sous forme d'un programme linéaire.*

Preuve : Pour chaque arête uv, remplaçons la première contrainte par :

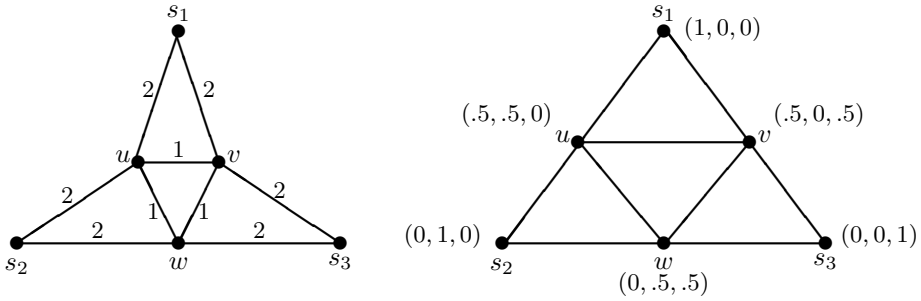
$$x_{uv}^i \geqslant x_u^i - x_v^i, \quad 1 \leqslant i \leqslant k$$

$$x_{uv}^i \geqslant x_v^i - x_u^i, \quad 1 \leqslant i \leqslant k$$

$$d(uv) = \frac{1}{2} \sum_{i=1}^{k} x_{uv}^i$$

Puisqu'on minimise la fonction objectif, toute solution optimale vérifie $x_{uv}^i = |x_u^i - x_v^i|$. Les autres contraintes sont clairement linéaires. □

Exemple 19.2 Voici un exemple où le coût de la coupe multiséparatrice fractionnaire optimale est strictement inférieur à celui de la coupe multiséparatrice entière optimale. L'association des sommets aux points de Δ_3 dans la solution optimale fractionnaire est illustrée ci-dessous : elle coûte 7.5. Quant à la solution entière optimale, elle coûte 8.



□

La propriété suivante va nous simplifier grandement la tâche :

Lemme 19.3 *Soit $\boldsymbol{x}$ une solution réalisable de la relaxation (19.1). Sans perte de généralité, nous pouvons supposer que pour toute arête $uv \in E$, les coordonnées de x_u et de x_v ne diffèrent qu'en au plus deux indices.*

Preuve : Nous allons découper les arêtes en rajoutant des sommets de façon à obtenir la propriété voulue sans modifier ni le coût de la solution, ni le coût optimal.

Donnons-nous une arête $uv \in E$ telle que les coordonnées de x_u et de x_v diffèrent en plus de deux indices. Remplaçons cette arête par deux nouvelles arêtes uw et wv, où w est un nouveau sommet. Attribuons le coût $c(uv)$ à chacune des nouvelles arêtes, pour ne pas modifier le coût optimal d'une solution entière. Voyons maintenant comment assurer que $d(uv) = d(uw) + d(wv)$, et donc que le coût de la solution fractionnaire reste inchangé.

Étudions les coordonnées où x_u et x_v diffèrent. Soit i une coordonnée où la différence est minimum. Quitte à échanger u et v, supposons que $x_u^i < x_v^i$. Posons $\alpha = x_v^i - x_u^i$. Il existe une coordonnée j telle que $x_u^j \geqslant x_v^j + \alpha$ (car $x_u, x_v \in \Delta_k$). Définissons le point x_w comme suit. Les i-ième et j-ième coordonnées de x_w sont : $x_w^i = x_u^i$ et $x_w^j = x_v^j + \alpha$. Les autres coordonnées de x_w sont les mêmes que celles de x_v. Clairement, $x_w \in \Delta_k$ et $d(uv) = d(uw) + d(wv)$.

Les coordonnées de v et de w ne diffèrent qu'en deux indices, et celles de u et de w diffèrent en moins d'indices que u et v. Ainsi, moins de $k-2$ telles divisions de chaque arête suffisent pour garantir la propriété voulue. □

19.2 Algorithme à base d'arrondi randomisé

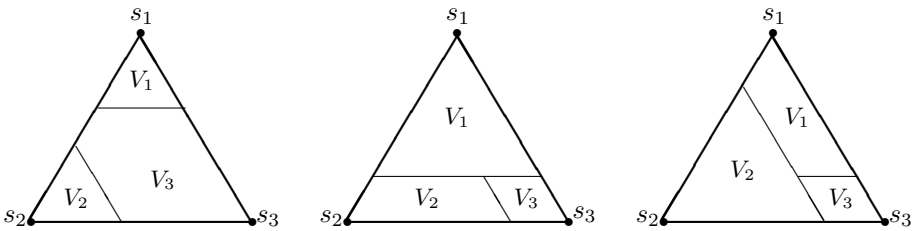
Soit $\boldsymbol{x}$ une solution optimale de la relaxation (19.1) vérifiant la propriété du lemme 19.3, et notons OPT_f son coût. Soit E_i l'ensemble des arêtes dont les i-ièmes coordonnées des extrémités diffèrent, c'est-à-dire $E_i = \{uv \in E : x_u^i \neq x_v^i\}$. Comme les sommets appartiennent à Δ_k, toute arête e telle que $d(e) > 0$, appartient à deux de ces ensembles exactement. Posons $W_i =$

$\sum_{e \in E_i} c(e)d(e)$ et réindexons les terminaux de sorte que W_k soit le maximum de $W_1, \ldots, W_k$. Posons, pour $\rho \in]0,1[$:

$$B(s_i, \rho) = \{v \in V \,:\, x_v^i \geqslant \rho\}.$$

Détaillons le fonctionnement l'algorithme 19.4. Il tire ρ uniformément dans $]0,1[$, et une permutation σ égale à $(1,2,\ldots,k-1,k)$ ou $(k-1,k-2,\ldots,1,k)$ avec probabilité $1/2$ chacune. ρ et σ lui servent à construire une partition de V en k ensembles $V_1, \ldots, V_k$ tels que $s_i \in V_i$. La coupe multiséparatrice sera constituée des arêtes entre ces ensembles.

Si σ est la première (resp., la seconde) permutation, alors ces ensembles sont construits dans l'ordre $V_1, V_2, \ldots, V_k$ (resp., $V_{k-1}, V_{k-2}, \ldots, V_1, V_k$). Si $\rho > 1/2$, les ensembles $B(s_i, \rho)$ sont deux à deux disjoints. Dans ce cas, la partition obtenue est indépendante de σ, car V_i est simplement $B(s_i, \rho)$ pour $1 \leqslant i \leqslant k-1$, et $V_k = V - (V_1 \cup \cdots \cup V_{k-1})$. Si $\rho \leqslant 1/2$, les ensembles $B(s_i, \rho)$ se recoupent et σ intervient ; la figure suivante illustre ce cas pour $k = 3$.



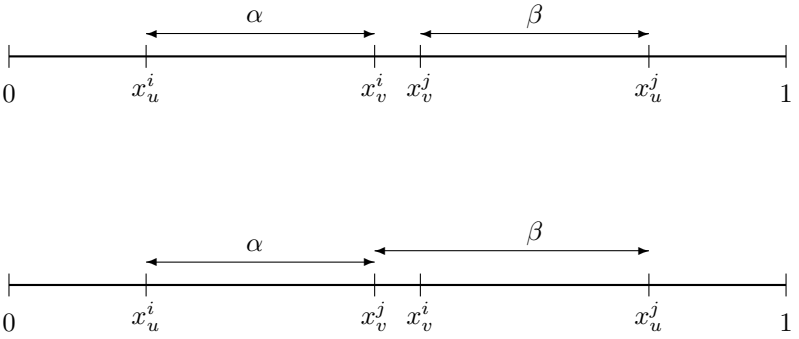
Algorithme 19.4 (Coupe multiséparatrice)

1. Calculer une solution optimale $\boldsymbol{x}$ de la relaxation (19.1).
2. Réindexer les terminaux pour que W_k soit le maximum de $W_1, \ldots, W_k$.
3. Tirer uniformément $\rho \in]0,1[$ et $\sigma \in \{(1,2,\ldots,k-1,k), (k-1,k-2,\ldots,1,k)\}$.
4. Pour $i = 1$ à $k-1$: $V_{\sigma(i)} \leftarrow B(s_{\sigma(i)}, \rho) - \bigcup_{j<i} V_{\sigma(j)}$.
5. $V_k \leftarrow V - \bigcup_{i<k} V_i$.
6. Soit C l'ensemble des arêtes entre les ensembles de la partition $V_1, \ldots, V_k$. Renvoyer C.

Nous allons démontrer que l'espérance du coût de la coupe multiséparatrice construite par l'algorithme, $\mathbf{E}[c(C)]$, est inférieure à $(1.5 - 1/k)\,\mathrm{OPT}_f$. Le lemme suivant est la clé de ce résultat.

Lemme 19.5 *Si $e \in E - E_k$, alors* $\mathbf{Pr}[e \in C] \leqslant 1.5\, d(e)$, *et si $e \in E_k$, alors* $\mathbf{Pr}[e \in C] \leqslant d(e)$.

Preuve : Supposons tout d'abord que $e \in E - E_k$. Posons $e = uv$ et notons i et j les indices des coordonnées où x_u et x_v diffèrent. Quitte à échanger i et j, $x_u^i < x_u^j$. Il y a deux cas : soit les intervalles $[x_u^i, x_v^i]$ et $[x_v^j, x_u^j]$ s'intersectent, soit ils sont disjoints. Ces deux cas sont illustrés ci-dessous. Remarquons que dans les deux cas, les intervalles ont la même longueur, car $x_v^i - x_u^i = x_u^j - x_v^j = d(e)$. La figure suivante définit les intervalles α et β dans les deux cas.



Les seuls cas où l'arête e peut être sélectionnée dans la coupe multiséparatrice, est celui où u et v sont placés dans deux ensembles différents parmi V_i, V_j, ou V_k. Remarquons que si $\rho \in [0,1] - (\alpha \cup \beta)$, alors les deux sommets seront placés dans le même ensemble, et l'arête ne fera pas partie de la coupe multiséparatrice. Clairement, $\mathbf{Pr}[\rho \in (\alpha \cup \beta)] = |\alpha| + |\beta| \leqslant 2d(e)$.

Le fait suivant est essentiel pour obtenir la majoration désirée : si $\rho \in \alpha$ et $\sigma(j) < \sigma(i)$, alors u et v seront placés tous les deux dans V_j, et e ne sera pas dans la coupe multiséparatrice. La probabilité de cet événement est clairement $|\alpha|/2$. Ainsi,

$$\mathbf{Pr}[e \in C] \leqslant |\beta| + |\alpha|/2 \leqslant 1.5\, d(e).$$

Reste à étudier le cas où $e \in E_k$, et où ses extrémités diffèrent sur les i-ième et k-ième coordonnées. Dans ce cas, $\sigma(i) < \sigma(k)$ et u et v seront placés dans des ensembles différents seulement si ρ tombe entre x_u^i et x_v^i. La probabilité d'un tel événement est $d(e)$. □

Lemme 19.6 *La coupe multiséparatrice C construite par l'algorithme 19.4 vérifie :*

$$\mathbf{E}[c(C)] \leqslant (1.5 - 1/k)\, \mathrm{OPT}_f\,.$$

Preuve : Par construction, C est une coupe multiséparatrice. Nous avons $\mathrm{OPT}_f = \sum_e c(e)d(e)$. Comme chaque arête de longueur non nulle appartient à deux des ensembles (E_i), $\sum_{i=1}^{k} W_i = 2\ \mathrm{OPT}_f$. Or, k est tel que W_k est le poids maximum de ces ensembles, donc

$$W_k = \sum_{e \in E_k} c(e)d(e) \geqslant \frac{2}{k}\ \mathrm{OPT}_f\,.$$

Ainsi,

$$\begin{aligned}
\mathbf{E}[c(C)] &= \sum_{e \in E} c(e)\mathbf{Pr}[e \in C] = \sum_{e \in E - E_k} c(e)\mathbf{Pr}[e \in C] + \sum_{e \in E_k} c(e)\mathbf{Pr}[e \in C] \\
&\leqslant 1.5 \sum_{e \in E - E_k} c(e)d(e) + \sum_{e \in E_k} c(e)d(e) \\
&= 1.5 \sum_{e \in E} c(e)d(e) - 0.5 \sum_{e \in E_k} c(e)d(e) \\
&\leqslant (1.5 - 1/k) \cdot \mathrm{OPT}_f
\end{aligned}$$

La première inégalité est une conséquence du lemme 19.5. □

Le lemme 19.6 majore le saut intégral de la relaxation 19.1 par $1.5 - 1/k$ (se reporter aux notes, section 19.5, pour des références sur une légère amélioration de ce résultat). La meilleure minoration connue du saut intégral est $8/(7 + \frac{1}{k-1})$; l'exemple 19.2 donne une minoration par 16/15.

La majoration de l'espérance du poids de la coupe multiséparatrice donnée au lemme 19.6 s'écrit également sous forme d'une inégalité avec forte probabilité de manière standard (voir exercices 1.10 et 19.4). Nous en concluons :

Théorème 19.7 *Le problème de la coupe multiséparatrice admet une* 3/2*-approximation randomisée.*

19.3 Demi-intégralité de la coupe de nœuds multiséparatrice

Le problème suivant est une généralisation du problème de la coupe multiséparatrice, dans le sens où il existe une réduction isofacteur du problème de la coupe multiséparatrice à celui-ci (voir exercice 19.13).

Problème 19.8 (Coupe de nœuds multiséparatrice) Étant donné un graphe connexe non orienté $G = (V, E)$ muni d'une fonction de coût sur les sommets $c : V \to \mathbf{R}^+$, et un ensemble indépendant de terminaux $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$ dans G, une *coupe de nœuds multiséparatrice* est un sous-ensemble de $V - S$, dont le retrait déconnecte les terminaux deux à deux. Le problème est de trouver une telle coupe de poids minimum.

Nous allons démontrer que la relaxation du problème linéaire en nombres entiers suivant admet toujours une solution optimale demi-entière. Nous en déduirons immédiatement une $(2-2/k)$-approximation (voir exercice 19.11). Dans ce programme, nous associons à chaque sommet $v \in V - S$ une variable d_v, à valeurs dans $\{0,1\}$, indiquant si v est sélectionné ou pas. Notons $\mathcal{P}$ l'ensemble de tous les chemins entre les différents terminaux. Nous avons une contrainte par chemin $p \in \mathcal{P}$ assurant qu'un sommet au moins est sélectionné sur ce chemin.

$$\begin{aligned} &\text{minimiser} && \sum_{v \in V-S} c_v d_v \\ &\text{sous les contraintes} && \sum_{v \in p} d_v \geqslant 1, && p \in \mathcal{P} \\ & && d_v \in \{0,1\}, && v \in V - S \end{aligned}$$

En voici la relaxation. Comme précédemment, d_v est interprété comme un indicateur de distance. Nous définissons la *longueur* d'un chemin, comme la somme des indicateurs de distance des sommets non terminaux sur ce chemin. La *distance* entre deux sommets sera la longueur du plus court chemin entre eux. Une solution $\boldsymbol{d}$ n'est réalisable que si la distance entre deux terminaux est toujours $\geqslant 1$.

$$\begin{aligned} &\text{minimiser} && \sum_{v \in V-S} c_v d_v && && (19.2) \\ &\text{sous les contraintes} && \sum_{v \in p} d_v \geqslant 1, && p \in \mathcal{P} \\ & && d_v \geqslant 0, && v \in V - S \end{aligned}$$

Comme au chapitre 18, le dual s'interprète comme un problème de multiflot. Les flots circulent entre les différents terminaux et la contrainte est que le flot total traversant chaque sommet est borné par le coût du sommet.

$$\begin{aligned} &\text{maximiser} && \sum_{p \in \mathcal{P}} f_p && && (19.3) \\ &\text{sous les contraintes} && \sum_{p\,:\,v \in p} f_p \leqslant c_v, && v \in V - S \\ & && f_p \geqslant 0, && p \in \mathcal{P} \end{aligned}$$

Soit $\boldsymbol{d}$ une solution optimale du programme (19.2). Voici comment obtenir efficacement une solution optimale demi-entière à partir de $\boldsymbol{d}$. Pour les besoins de la preuve uniquement, considérons $\boldsymbol{f}$ une solution optimale du programme dual. Les conditions des écarts complémentaires donnent :

Conditions primales : Pour tout $v \in V - S$, si $d_v > 0$ alors v est saturé.

Conditions duales : Pour tout chemin p, si $f_p > 0$ alors la longueur de p est exactement 1.

Considérons le graphe G avec ses indicateurs de distances d_v sur chaque sommet $v \in V - S$. Nous définissons la *région* S_i de tout terminal s_i, comme l'ensemble des sommets accessibles depuis s_i par des chemins de longueur nulle (par définition, $s_i \in S_i$). La *frontière* B_i de S_i est l'ensemble des sommets adjacents et extérieurs à S_i, c'est-à-dire $B_i = \{v \in \overline{S_i} : \exists u \in S_i, uv \in E\}$. Comme $\boldsymbol{d}$ est réalisable, les k régions sont disjointes et leurs frontières ne contiennent aucun terminal.

Fait 19.9 *S'il existe $i \neq j$ tels que $v \in B_i \cap B_j$, alors $d_v = 1$.*

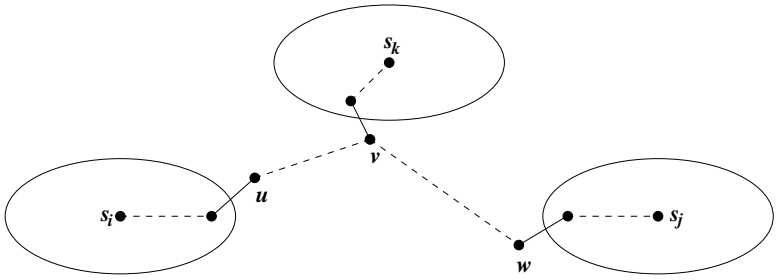
Preuve : Comme $v \in B_i \cap B_j$, il existe un chemin de s_i à s_j sur lequel v est le seul sommet d'indicateur de distance non nul. Comme $\boldsymbol{d}$ est réalisable et optimale, $d_v = 1$. □

Soit $M = \bigcup_{i=1}^{k} B_i$ l'ensemble des sommets sur les frontières. Partitionnons M en deux ensembles : M^{disj} l'ensemble des sommets qui appartiennent à une seule frontière, et M^{int} les autres. D'après le fait 19.9, l'indicateur de distance de tout sommet de M^{int} vaut 1.

Lemme 19.10 *Soit p un chemin entre deux terminaux distincts tel que $f_p > 0$. Alors, depuis tout sommet de M, p passe par exactement un sommet de M^{int} ou exactement deux sommets de M^{disj}.*

Preuve : D'après le théorème des écarts complémentaires, la longueur de p est exactement 1. Ainsi, si p passe par un sommet de M^{int}, il ne passe par aucun autre sommet de M.

Supposons par l'absurde que p passe par trois sommets de M^{disj} ou plus. Soient s_i le début et s_j la fin de p, et u et w les premier et dernier sommets de M^{disj} sur p, respectivement. Soit v un sommet intermédiaire de M^{disj} sur p. Puisque $v \in M^{\text{disj}}$, v appartient à une unique frontière B_l ; remarquons qu'il est possible que $l = i$ ou $l = j$.



Soit q un chemin reliant v à s_l par des sommets de S_l (un tel chemin existe car $v \in B_l$). Étudions les deux chemins suivants : le premier est la partie de p allant de s_i à v suivie de q, et le second est le chemin q renversé suivi de la partie de p allant de v à s_j. Les extrémités d'au moins l'un de ces deux chemins sont deux terminaux distincts (même si $l = i$ ou $l = j$). Or, puisque ce chemin manque au moins un des sommets d'indicateur non nul de p, sa longueur est strictement inférieure à 1, contradiction car $\boldsymbol{d}$ est réalisable. CQFD. □

Soit $\boldsymbol{h}$ la solution du programme (19.2) définie à partir de $\boldsymbol{d}$ sur tout sommet v par : $h_v = 1$ si $v \in M^{\text{int}}$; $h_v = 1$ si $v \in M^{\text{disj}}$; et $h_v = 0$ sinon.

Lemme 19.11 *$\boldsymbol{h}$ est une solution optimale du programme (19.2).*

Preuve : Tout chemin valide p entre deux terminaux s_i et s_j passe par des sommets sur les frontières B_i et B_j. Supposons qu'un tel chemin passe par $v \in B_i \cap B_j$. Par définition, $v \in M^{\text{int}}$, et donc $h_v = 1$. Sinon, il passe par deux sommets de M^{disj}. Dans les deux cas, la longueur de p est 1 et $\boldsymbol{h}$ est donc réalisable.

Prouvons maintenant que $\boldsymbol{h}$ est optimale en démontrant que sa valeur objectif est celle du flot $\boldsymbol{f}$. Partitionnons les chemins transportant un flot non nul dans $\boldsymbol{f}$ en deux catégories : $\mathcal{P}_1$ est l'ensemble des chemins passant par un sommet de M^{int} et $\mathcal{P}_2$ est l'ensemble des chemins passant par deux sommets de M^{disj}. D'après le lemme 19.10, ce sont les deux seules possibilités. D'après le théorème des écarts complémentaires et l'optimalité de $\boldsymbol{d}$, tout sommet de M est saturé par $\boldsymbol{f}$. Ainsi, le flot total charrié par les chemins de $\mathcal{P}_1$ est $\sum_{v \in M^{\text{int}}} c_v$ et celui qui est charrié par les chemins de $\mathcal{P}_2$ est $\frac{1}{2} \sum_{v \in M^{\text{disj}}} c_v$. Le flot total vaut donc :

$$\sum_{v \in M^{\text{int}}} c_v + \frac{1}{2} \sum_{v \in M^{\text{disj}}} c_v = \sum_{v \in V - S} h_v c_v.$$

CQFD. □

$\boldsymbol{h}$ s'obtient simplement en temps polynomial à partir de toute solution optimale $\boldsymbol{d}$ du programme (19.2), ainsi :

Théorème 19.12 *Le programme (19.2) admet toujours une solution optimale demi-entière. De plus, on peut construire en temps polynomial une solution optimale demi-entière à partir de toute solution optimale.*

19.4 Exercices

Nous avons présenté au chapitre 4 une $(2 - 2/k)$-approximation pour le problème de la coupe multiséparatrice de poids minimum en comparant la

solution trouvée à une solution optimale entière. Les deux exercices suivants proposent un algorithme ayant la même garantie à base de dualité.

19.1 Étant donné des terminaux $s_1, \ldots, s_k$, considérons le problème de multiflot où les paires de terminaux forment les paires source-puits (les coûts des sommets sont leurs capacités). Il y a $\binom{k}{2}$ flots. Proposez un programme linéaire maximisant ce flot et déterminez son dual. Le dual recherche des indicateurs de distance sur les arêtes qui vérifient l'inégalité triangulaire et garantissent que la distance entre deux terminaux quelconques est supérieure à 1. Une solution optimale du dual est une coupe multiséparatrice fractionnaire.

19.2 Étudions l'algorithme suivant pour trouver une coupe multiséparatrice. Trouver une coupe multiséparatrice fractionnaire, solution optimale du programme dual. À cette solution correspondent des indicateurs de distances d. Tirer ρ uniformément dans $[0, \frac{1}{2}]$. Chaque arête uv est sélectionnée ssi il existe un terminal s tel que $d(u,s) \leqslant \rho \leqslant d(v,s)$. Démontrez que l'espérance du coût de la coupe multiséparatrice obtenue est inférieure au double du coût de la coupe multiséparatrice fractionnaire optimale. Dérandomisez cet algorithme. Modifiez-le pour en faire une $(2 - 2/k)$-approximation.
Indication : Démontrez que la probabilité de sélectionner une arête uv est majorée par $2\,d(u,v)$.

19.3 Afin de tenter d'améliorer l'algorithme précédent, tirons ρ uniformément dans $[0,1]$. Pourquoi est-ce inopérant ? Comment cela est-il corrigé dans l'algorithme 19.4 ?

19.4 Déduisez le théorème 19.7 du lemme 19.6.
Indication : Le lemme 19.6 donne que $\mathbf{Pr}[c(C) \leqslant 1.5 \cdot \mathrm{OPT}_f] \geqslant 2/k \geqslant 2/n$. Exécutez l'algorithme 19.4 un nombre polynomial de fois et renvoyez la meilleure coupe multiséparatrice.

19.5 Que devient le facteur d'approximation de l'algorithme 19.4 si σ est une permutation aléatoire de $\mathfrak{S}_k$?

19.6 (Y. Rabani) Lorsque $k = 3$, remplacez l'étape d'arrondi aléatoire de l'algorithme 19.4 par ce qui suit. Tirer ρ_1 et ρ_2 indépendamment et uniformément dans $]0,1[$. Sélectionner l'une des trois dimensions i uniformément. Regrouper avec s_i tous les sommets non terminaux tels que $x_v^i \geqslant \rho_1$. Sélectionner arbitrairement une des deux autres dimensions, j ; noter la dernière dimension l. Regrouper avec s_j les sommets restants non terminaux tels que $x_v^j + x_v^i/2 \geqslant \rho_2$. Enfin, regrouper avec s_k les sommets non terminaux restants. Démontrez que cet algorithme modifié est une 7/6-approximation pour le problème de la coupe multiséparatrice à trois terminaux.

19.7 Nous proposons dans cet exercice une autre relaxation du problème de la coupe multiséparatrice (voir également chapitre 30). Étant donné un

graphe non orienté $G = (V, E)$ muni de coûts sur les arêtes, nous lui associons le graphe orienté H obtenu en remplaçant chaque arête uv de G par deux arcs $(u \to v)$ et $(v \to u)$, ayant le même coût que uv. Associons à chaque arc e de H, un indicateur d_e à valeurs dans $\{0, 1\}$. Numérotons arbitrairement les terminaux $s_1, \ldots, s_k$. Notons $\mathcal{P}$ l'ensemble des chemins simples allant d'un terminal de numéro inférieur à un autre terminal de numéro supérieur. Étudions le programme linéaire entier *orienté* pour le problème de la coupe multiséparatrice.

$$\begin{aligned} &\text{minimiser} && \sum_{e \in H} c(e) d_e && (19.4)\\ &\text{sous les contraintes} && \sum_{e \in p} d_e \geqslant 1, \qquad p \in \mathcal{P} \\ & && d_e \in \{0, 1\}, \qquad e \in H \end{aligned}$$

1. Démontrez qu'à toute solution optimale du programme en nombres entiers (19.4) correspond une solution optimale au problème de la coupe multiséparatrice.
2. Exhibez une relaxation du programme et son dual. Proposez une interprétation « physique » du dual.
3. Montrez que le graphe de l'exemple 19.2 a un saut intégral de 16/15 pour cette relaxation également (en démontrant que les solutions primale et duale coûtent 7.5).

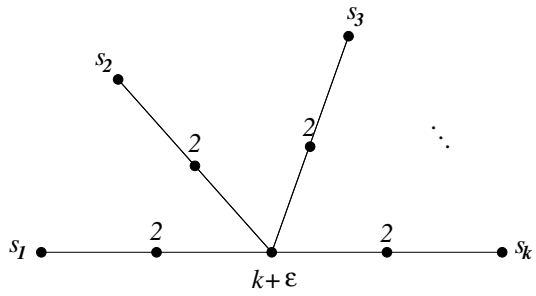
19.8 Étudions l'algorithme 4.3 pour le problème de la coupe multiséparatrice. Démontrez que l'algorithme analogue pour le problème de la coupe de nœuds multiséparatrice, fondé sur des coupes séparatrices, ne conduit pas à un facteur d'approximation constant. Quel est le meilleur facteur que vous puissiez montrer pour cet algorithme ?

19.9 Le problème de la coupe multiséparatrice a aussi la propriété de demi-intégralité. Démontrez ce fait en exhibant un programme linéaire pour le problème de la coupe multiséparatrice, similaire au programme (19.2).

19.10 Montrez que le minorant de OPT donné par le programme (19.2) peut être $2 - 2/k$ fois inférieur à OPT, en exhibant un graphe dans lequel la coupe de nœuds multiséparatrice optimale est $2 - 2/k$ fois plus grande que le flot maximum.

19.11 Le théorème 19.12 donne directement une 2-approximation pour le problème de la coupe de nœuds multiséparatrice, en arrondissant vers le haut les demis dans toute solution optimale demi-entière. Proposez une $(2 - 2/k)$-approximation et exhibez une famille d'instances critiques pour cet algorithme.

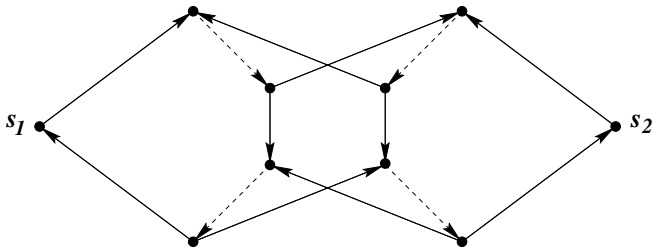
Indication : Tous les sommets de M^{disj} ne sont pas nécessaires pour obtenir une coupe multiséparatrice. Étudiez le graphe suivant pour vos instances critiques.



19.12 Étudions le problème suivant.

Problème 19.13 (Coupe multiséparatrice orientée)[3] Étant donné un graphe orienté $G = (V, E)$ muni de capacités $c : E \to \mathbf{R}^+$ sur ses arcs, et un ensemble de terminaux $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, une *coupe multiséparatrice orientée* est un ensemble d'arcs dont le retrait coupe tous les chemins allant d'un terminal à un autre. Le problème est de trouver une telle coupe de poids minimum.

Proposez une relaxation d'un programme linéaire similaire à (19.2) pour ce problème. Le dual s'interprète comme le programme d'un multiflot orienté. Déterminez la coupe multiséparatrice fractionnaire optimale et le flot optimal pour le graphe suivant :



Remarquez que contrairement au programme (19.2), cette relaxation n'admet pas toujours de solutions optimales demi-entières.

19.13 Considérons les deux problèmes suivants :

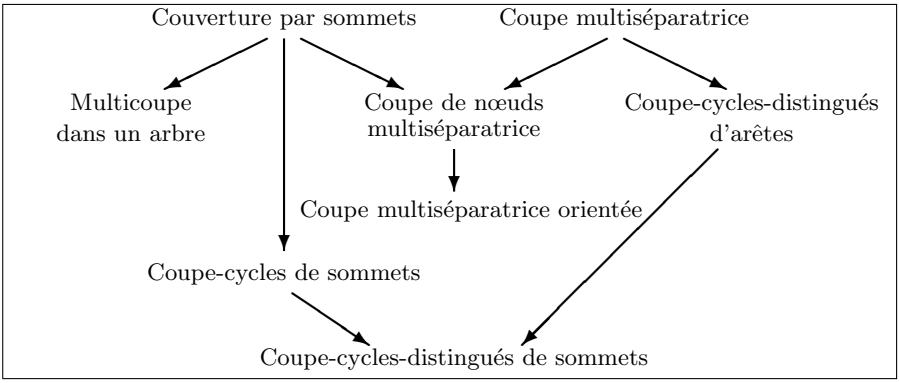
Problème 19.14 (Coupe-cycles-distingués d'arêtes)[4] Étant donné un graphe non orienté connexe $G = (V, E)$ muni d'une fonction de poids

[3] *Directed multiway cut*, en anglais.
[4] *Subset feedback edge set*, en anglais.

sur les arêtes $w : E \to \mathbf{R}^+$, et un ensemble de sommets distingués $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, un *coupe-cycles-distingués d'arêtes* est un ensemble d'arêtes dont le retrait élimine tous les cycles du graphe passant par un des sommets distingués. Le problème est de trouver un tel sous-ensemble de poids minimum.

Problème 19.15 (Coupe-cycles-distingués de sommets)[5] Étant donné un graphe non orienté connexe $G = (V, E)$ muni d'une fonction de poids sur les sommets $w : V \to \mathbf{R}^+$, et un ensemble de sommets distingués $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, un *coupe-cycles-distingués de sommets* est un sous-ensemble de $V - S$ dont le retrait du graphe élimine tous les cycles passant par un des sommets distingués. Le problème est de trouver un tel sous-ensemble de poids minimum.

Ces problèmes et ceux introduits précédemment sont apparentés par des réductions isofacteurs illustrées sur la figure ci-dessous (chaque arête représente une telle réduction). Donnez ces réductions (se reporter à la section A.3.1 pour une définition de ce type de réduction).



Les meilleurs facteurs d'approximation connus à ce jour pour les problèmes de la coupe multiséparatrice et du coupe-cycles-distingués de sommets sont 1.34 et 8, respectivement. Pour les autres problèmes, le meilleur facteur connu actuellement est 2.

19.5 Notes

L'algorithme 19.4 est dû à Calinescu, Karloff et Rabani [38]. Le meilleur facteur d'approximation connu pour le problème de la coupe multiséparatrice

[5] *Subset feedback vertex set*, en anglais.

est 1.3438, obtenu par Karger, Klein, Stein, Thorup et Young [165]. C'est aussi la meilleure majoration connue du saut intégral pour la relaxation utilisée. Dans [94], Freund et Karloff proposent une famille d'instances minorant ce saut intégral par $8/(7 + \frac{1}{k-1})$; l'exemple 19.2 est tiré de cet article. Le théorème 19.12 est dû à Garg, Vazirani et Yannakakis [103]. Se reporter à Naor et Zosin [219], Even, Naor, Schieber et Zosin [83] et Even, Naor et Zosin [84], pour les meilleures approximations connues à ce jour pour, respectivement, les problèmes de la coupe multiséparatrice orientée, du coupe-cycles-distingués d'arêtes et de sommets.

20 Multicoupe dans les graphes

Nous avons mentionné au chapitre 1 l'importance des relations min-max en optimisation combinatoire. La plus utile d'entre elles est sans doute le célèbre théorème de la coupe minimum et du flot maximum. En effet, une grande partie de la théorie des flots et de celle des coupes dans les graphes repose sur ce théorème. Il n'est donc pas surprenant que des efforts considérables aient été menés pour obtenir des généralisations de ce résultat au cas des multiflots.

Il en existe deux généralisations. La première consiste à maximiser la somme des flots transportés, étant donné les contraintes de conservation de flot et de capacité. Dans la seconde, $\mathrm{dem}(i)$ unités de chaque flot i sont demandées, et le but est de maximiser le débit[1] f, tel que pour tout i, $f \cdot \mathrm{dem}(i)$ unités du flot i puissent être acheminées simultanément. Nous appellerons respectivement ces deux problèmes le *multiflot total maximum*[2] et le *multiflot sur demande*.[3] Clairement, lorsqu'il n'y a qu'un seul flot, ces deux problèmes sont celui du flot maximum.

Ces deux généralisations sont associées à deux problèmes de coupes **NP**-difficiles : le premier avec le problème de la multicoupe minimum (problème 18.1), et le second avec le problème de la coupe la moins dense[4] (problème 21.2). Dans les deux cas, un théorème de flot maximum–coupe minimum approché se déduit de tout algorithme d'approximation pour le problème de coupe associé. Dans ce chapitre, nous étudions la première généralisation, et le chapitre 21 présentera la seconde. Nous donnons une $O(\log k)$-approximation pour le problème de la multicoupe, où k est le nombre de terminaux. Le chapitre 18 a présenté une 2-approximation pour le cas particulier où le graphe est un arbre.

[1] *Throughput*, en anglais.
[2] *Sum multicommodity flow problem*, en anglais.
[3] *Demands multicommodity flow problem*, en anglais.
[4] *Sparsest cut problem*, en anglais.

20.1 Multiflot total maximum

Problème 20.1 (Multiflot total maximum)[5] Soit un graphe $G = (V, E)$ non orienté où chaque arête $e \in E$ a une capacité c_e. On se donne un ensemble de paires de sommets toutes distinctes $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ (les sommets des différentes paires ne sont pas nécessairement distincts). Un flot particulier est associé à chaque paire (s_i, t_i), où s_i est la source du flot et t_i son puits. Le but est de maximiser la somme des flots routés. Les contraintes sont que chaque flot est conservé en tous les sommets (autres que sa source et son puits) et que la somme de tous les flots traversant une arête dans n'importe quel sens, doit être inférieure à la capacité de l'arête.

Commençons par exprimer ce problème sous forme d'un programme linéaire. Pour chaque flot i, notons P_i l'ensemble des chemins de s_i à t_i dans G, et posons $P = \bigcup_{i=1}^{k} P_i$. Associons à chaque $p \in P$, une variable f_p quantifiant le flot qui passe par le chemin p. La nature du flot passant par ce chemin est complètement déterminée par les extrémités du chemin. Le but est de maximiser la somme des flots passant par ces chemins étant donné les contraintes de capacités sur les arêtes. Remarquons que cette formulation assure que les flots se conservent en chaque sommet. Le programme a un nombre exponentiel de variables ; ceci n'a pas d'importance, car nous ne l'utiliserons essentiellement que pour obtenir une formulation simple du dual.

$$\begin{aligned} &\text{maximiser} && \sum_{p \in P} f_p && && (20.1)\\ &\text{sous les contraintes} && \sum_{p\,:\,e \in p} f_p \leqslant c_e, && e \in E \\ & && f_p \geqslant 0, && p \in P \end{aligned}$$

En voici le programme dual. Le dual associe à chaque arête e, une variable d_e, que nous interprétons comme un indicateur de distance.

$$\begin{aligned} &\text{minimiser} && \sum_{e \in E} c_e d_e && && (20.2)\\ &\text{sous les contraintes} && \sum_{e \in p} d_e \geqslant 1, && p \in P \\ & && d_e \geqslant 0, && e \in E \end{aligned}$$

Le programme dual recherche des indicateurs de distance, tels que la somme des indicateurs le long de tout chemin $p \in P$ soit supérieure à 1. De manière équivalente, une instanciation des indicateurs de distance est réalisable ssi

[5] *Sum multicommodity flow problem*, en anglais.

pour tout flot i, la longueur du chemin le plus court de s_i à t_i (relativement aux indicateurs) est supérieure à 1.

Remarquons que les programmes (18.2) et (18.1) sont les cas particuliers des deux programmes ci-dessus où G est un arbre.

Les remarques suivantes, faites au chapitre 18, sont toujours valables : toute solution optimale entière du programme (20.2) définit une multicoupe minimum, et toute solution fractionnaire optimale s'interprète comme une multicoupe fractionnaire minimum. D'après le théorème de dualité en programmation linéaire, le poids de la multicoupe fractionnaire minimum est égale à la valeur du multiflot maximum. L'exemple 18.2 montre cependant qu'il peut être strictement inférieur au poids de la multicoupe entière minimum.

Se pose alors naturellement la question de savoir si le rapport entre le poids d'une multicoupe minimum et la valeur du multiflot maximum est borné. Le saut intégral du programme (20.2) est-il majoré ? La section suivante présente un algorithme qui trouve une multicoupe à un facteur $O(\log k)$ du flot maximum, démontrant ainsi que le saut intégral est en $O(\log k)$.

20.2 Algorithme à base d'arrondi

Commençons par remarquer que le programme dual (20.2) se résout en temps polynomial par l'algorithme des ellipsoïdes (voir par exemple, [123]), car il est très simple d'obtenir un oracle séparateur. Il suffit de calculer la longueur du chemin le plus court de s_i à t_i pour chaque flot i, relativement aux indicateurs de distances courants. Si toutes ces longueurs sont supérieures à 1, la solution est réalisable. Sinon, l'inégalité correspondant au plus court de ces chemins est violée. L'exercice 20.1 donne un programme alternatif, résoluble en temps polynomial. Notons d_e l'indicateur de distance calculé pour chaque arête e, et posons $F = \sum_{e \in E} c_e d_e$.

Notre but est de sélectionner des arêtes de petites capacités, relativement à F, pour en faire notre multicoupe. Notons D l'ensemble des arêtes d'indicateurs de distance non nuls, c'est-à-dire $D = \{e \mid d_e > 0\}$. Clairement, D est une multicoupe ; cependant, sa capacité peut être très grande par rapport à F (voir exercices 20.3 et 20.4). Comment sélectionner un sous-ensemble de D de faible capacité, qui soit encore une multicoupe ? Comme la multicoupe fractionnaire optimale est la façon la plus efficace de déconnecter tous les paires source-puits, les arêtes qui reçoivent les plus grands indicateurs de distances sont plus importantes que celles qui en reçoivent des plus petits. L'algorithme ci-dessous favorise indirectement les arêtes ayant de grands indicateurs de distance.

L'algorithme fonctionne également sur un graphe $G = (V, E)$ où la longueur de chaque arête e est donnée par la fonction d_e. Le *poids d'une arête* e est défini par $c_e d_e$. Notons $\text{dist}(u, v)$ la longueur du chemin le plus court entre u et v dans ce graphe. Pour tout ensemble de sommets $S \subset V$, notons $\delta(S)$

l'ensemble des arêtes de la coupe $(S, \overline{S})$, puis $c(S)$ la capacité de cette coupe, c'est-à-dire la somme des capacités des arêtes de $\delta(S)$, et enfin poids(S) le *poids* de l'ensemble S, qui est *grosso modo* la somme des poids des arêtes dont les deux extrémités appartiennent à S (la définition précise sera donnée par la suite).

L'algorithme construit des *régions* dans G, c'est-à-dire des ensembles disjoints de sommets $S_1, \ldots, S_l$ tels que :

- Pour toute paire (s_i, t_i), s_i et t_i n'appartiennent pas à la même région, et au moins l'un des deux appartient à l'une des régions.
- Pour toute région S_i, $c(S_i) \leqslant \varepsilon$ poids(S_i), où le paramètre ε sera défini plus tard.

D'après la première condition, l'union des coupes définies par ces régions $M = \delta(S_1) \cup \delta(S_2) \cup \ldots \cup \delta(S_l)$ est une multicoupe, et d'après la seconde condition, sa capacité $c(M)$ est $\leqslant \varepsilon F$ (cette inégalité sera légèrement modifiée lorsque nous préciserons la définition de poids(S)).

20.2.1 Expansion d'une région par un processus continu

Les ensembles $S_1, \ldots, S_l$ sont construits par un processus d'expansion. Nous commençons par un processus continu qui présente plus clairement l'approche. Pour des raisons d'efficacité, l'algorithme utilisera, quant à lui, un processus discret (voir section 20.2.2).

Chaque région est obtenue par extension d'un ensemble à partir d'un sommet qui est la source ou le puits d'une paire. Ce sommet sera appelé la *racine* de la région. Supposons que la racine soit s_1. Le processus consiste à agrandir une boule autour de la racine. Notons $S(r)$ l'ensemble de sommets à distance $\leqslant r$ de s_1, c'est-à-dire $S(r) = \{v \mid \text{dist}(s_1, v) \leqslant r\}$. Nous avons $S(0) \supseteq \{s_1\}$, et quand r augmente continûment à partir de zéro, $S(r)$ s'étend par ajout de sommets par ordre de distance croissante à s_1, pour certaines valeurs discrètes de r.

Lemme 20.2 *Si le processus d'expansion des régions se termine avant que le rayon r atteigne $1/2$, alors l'ensemble S construit ne contient aucune paire source-puits.*

Preuve : La distance entre deux sommets de $S(r)$ est $\leqslant 2r$. Or, pour chaque flot i, dist$(s_i, t_i) \geqslant 1$, CQFD. □

Pour des raisons techniques qui apparaîtront au lemme 20.3 (voir aussi les exercices 20.5 et 20.6), nous attribuons un poids à la racine : poids$(s_1) = F/k$. Intuitivement, le poids de $S(r)$ est la somme de poids(s_1) et des poids de toutes les arêtes, ou parties d'arêtes, dans la boule de rayon r autour de s_1. Formellement, notons q_e la *proportion* appartenant à $S(r)$ de chaque arête e ayant (au moins) une extrémité dans $S(r)$. Si les deux extrémités de e sont dans $S(r)$, alors $q_e = 1$. Sinon, $e = uv$ avec $u \in S(r)$ et $v \notin S(r)$, et

$$q_e = \frac{r - \mathrm{dist}(s_1, u)}{\mathrm{dist}(s_1, v) - \mathrm{dist}(s_1, u)}.$$

Le *poids de la région* $S(r)$ est alors défini par

$$\mathrm{poids}(S(r)) = \mathrm{poids}(s_1) + \sum_{e\,:\,e \cap S(r) \neq \varnothing} c_e d_e q_e.$$

où la somme est prise sur les arêtes ayant au moins une extrémité dans $S(r)$.

Nous recherchons ε tel qu'on puisse garantir que $c(S(r)) \leqslant \varepsilon\ \mathrm{poids}(S(r))$ pour $r < 1/2$. L'observation importante est qu'à chaque instant, le poids de la région augmente à une vitesse $\geqslant c(S(r))$. Tant que $c(S(r)) > \varepsilon \cdot \mathrm{poids}(S(r))$,

$$\mathrm{dpoids}(S(r)) \geqslant c(S(r))\ \mathrm{d}r > \varepsilon\ \mathrm{poids}(S(r))\ \mathrm{d}r.$$

L'exercice 20.5 permettra au lecteur de mieux comprendre le processus.

Lemme 20.3 *La valeur* $\varepsilon = 2\ln(k+1)$ *suffit pour garantir qu'il existe* $r < 1/2$ *vérifiant* $c(S(r)) \leqslant \varepsilon\ \mathrm{poids}(S(r))$.

Preuve : Supposons, par l'absurde, que tout au long du processus d'expansion de la région, démarrant avec $r = 0$ et finissant avec $r = 1/2$, $c(S(r)) > \varepsilon\ \mathrm{poids}(S(r))$. À chaque instant, la variation infinitésimale du poids de la région est

$$\partial\mathrm{poids}(S(r)) = \sum_e c_e d_e\, \partial q_e.$$

Seules les arêtes ayant une extrémité seulement dans $S(r)$ contribuent au poids. Considérons une telle arête $e = uv$ avec $u \in S(r)$ et $v \notin S(r)$. Alors,

$$c_e d_e\, \partial q_e \;=\; c_e\, \frac{d_e}{\mathrm{dist}(s_1, v) - \mathrm{dist}(s_1, u)}\ \partial r.$$

Puisque $\mathrm{dist}(s_1, v) \leqslant \mathrm{dist}(s_1, u) + d_e$, nous avons $d_e \geqslant \mathrm{dist}(s_1, v) - \mathrm{dist}(s_1, u)$, et donc $c_e d_e\, \partial q_e \geqslant c_e\, \partial r$. D'où,

$$\partial\mathrm{poids}(S(r)) \geqslant c(S(r))\, \partial r > \varepsilon\ \mathrm{poids}(S(r))\, \partial r.$$

Ainsi, jusqu'à ce que la condition de terminaison soit vérifiée, le poids de la région augmente exponentiellement avec le rayon. Le poids initial de la région est F/k, et le poids final est inférieur à $F + F/k$. En intégrant, nous obtenons

$$\int_{\frac{F}{k}}^{F+\frac{F}{k}} \frac{1}{\mathrm{poids}(S(r))}\ \partial\mathrm{poids}(S(r)) > \int_0^{\frac{1}{2}} \varepsilon\, \mathrm{d}r.$$

Et donc, $\ln(k+1) > \frac{1}{2}\varepsilon$. D'où une contradiction avec $\varepsilon = 2\ln(k+1)$, CQFD. □

20.2.2 Le processus d'expansion discret

Le processus d'expansion discret démarre avec $S = \{s_1\}$ et ajoute les sommets à S dans l'ordre croissant de leurs distances respectives à s_1 (il s'agit essentiellement de calculer le chemin le plus court de chaque sommet à la racine). Clairement, les ensembles calculés par les deux processus sont identiques.

Nous redéfinissons le poids de la région S pour le processus discret comme suit :

$$\text{poids}(S) = \text{poids}(s_1) + \sum_{e\,:\,e\cap S\neq\varnothing} c_e d_e,$$

où la somme est prise sur les arêtes ayant au moins une extrémité dans S, et $\text{poids}(s_1)$ est fixé à F/k (l'exercice 20.6 justifiera le choix de $\text{poids}(s_1) = F/k$). Le processus discret s'arrête dès que $c(S) \leqslant \varepsilon\ \text{poids}(S)$, où $\varepsilon = 2\ln(k+1)$. Remarquons que pour le même ensemble S, $\text{poids}(S)$ est, dans le processus discret, supérieur à celui du processus continu. Par conséquent, le processus discret termine avec un ensemble plus petit que celui du processus continu. L'ensemble S construit ne contient donc aucune paire source-puits.

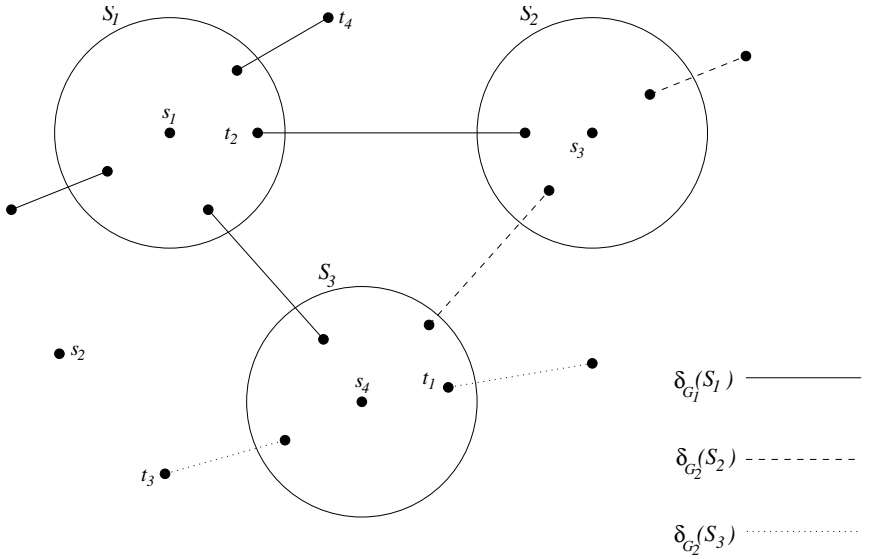
20.2.3 Construction itérative des régions

La première région est construite dans le graphe G, en prenant n'importe pour racine une source quelconque. Les autres régions sont construites l'une après l'autre. Posons $G_1 = G$ et notons S_1 la région construite dans G_1. La i-ième itération de l'algorithme commence quand les régions $S_1, \ldots, S_{i-1}$ sont construites. Notons G_i le sous-graphe de G induit par les sommets de $V - (S_1 \cup \ldots \cup S_{i-1})$.

Si G_i ne contient aucune paire source-puits, l'algorithme est terminé. Sinon, prenons la source s_j d'une telle paire pour racine, donnons-lui pour poids F/k, et construisons sa région dans G_i par expansion. Toutes les notions, telles que distance et poids, sont maintenant définies relativement au graphe G_i. Nous les indexons par G_i. Ainsi, pour l'ensemble des sommets S de G_i, $c_{G_i}(S)$ est la capacité totale des arêtes incidentes à S dans G_i, c'est-à-dire la somme des capacités des arêtes de $\delta_{G_i}(S)$. Comme précédemment, $\varepsilon = 2\ln(k+1)$, et la condition d'arrêt de l'expansion est $c_{G_i}(S_i) \leqslant \varepsilon\ \text{poids}_{G_i}(S_i)$. Remarquons que seule la racine a un poids non nul durant chaque itération.

L'algorithme construit les régions $S_1, \ldots, S_l$, $l \leqslant k$, puis renvoie l'ensemble $M = \delta_{G_1}(S_1) \cup \ldots \cup \delta_{G_l}(S_l)$. Puisque les arêtes de chaque coupe sont ôtées

du graphe dans les itérations suivantes, les coupes sont disjointes, et $c(M) = \sum_i c_{G_i}(S_i)$.



L'algorithme est présenté en encadré ci-dessous. Remarquons que du fait de l'expansion des régions, les arêtes ayant de grands indicateurs de distances, restent dans la coupe plus longtemps, et donc ont plus de chance d'être sélectionnées dans la multicoupe renvoyée. Bien sûr, la durée précise de la présence d'une arête dans la coupe est donnée par la différence des distances à la racine de ses deux extrémités. Comme annoncé, l'algorithme favorise donc indirectement les arêtes ayant de grands indicateurs de distance.

Algorithme 20.4 (Multicoupe minimum)

1. Calculer une solution optimale du programme (20.2), et donc les indicateurs de distances sur les arêtes de G.
2. $\varepsilon \leftarrow 2\ln(k+1)$, $H \leftarrow G$, $M \leftarrow \varnothing$.
3. Tant qu'il existe une paire source-puits dans H faire :
   Prendre la source s_j d'une telle paire.
   Étendre la région S de racine s_j jusqu'à ce que $c_H(S) \leqslant \varepsilon$ $\text{poids}_H(S)$.
   $M \leftarrow M \cup \delta_H(S)$.
   $H \leftarrow$ le sous-graphe de H, induit par l'élimination des sommets de S.
4. Renvoyer M.

Lemme 20.5 *L'ensemble M renvoyé est une multicoupe.*

Preuve : Nous devons démontrer qu'aucune des régions ne contient de paires source-puits. Durant la i-ième itération, la somme du poids de la racine courante et des poids des arêtes du graphe est majorée par $F+F/k$. D'après la preuve du lemme 20.3, le processus continu d'expansion de la région satisfait la condition d'arrêt avant que le rayon de la région n'atteigne $1/2$. Ainsi, la distance entre deux sommets de la région S_i construite par le processus discret, est également inférieure à 1. Remarquons que nous avions défini ces distances relativement au graphe G_i. Comme G_i est un sous-graphe de G, la distance entre deux sommets dans G est nécessairement inférieure à celle dans G_i. Ainsi, S_i ne contient aucune paire source-puits. □

Lemme 20.6 $c(M) \leqslant 2\varepsilon F = 4\ln(k+1)F$.

Preuve : Durant la i-ième itération, la condition d'arrêt donne $c_{G_i}(S_i) \leqslant \varepsilon \text{ poids}_{G_i}(S_i)$. Comme toutes les arêtes contribuant au $\text{poids}_{G_i}(S_i)$ sont ôtées du graphe à la fin de cette itération, chaque arête de G contribue au poids d'une région au plus. Le poids total des arêtes de G est F. Puisque chaque itération déconnecte au moins une paire source-puits, le nombre d'itérations est inférieur à k. Le poids total attribué aux racines est donc inférieur à F. Nous obtenons donc par sommation :

$$c(M) = \sum_i c_{G_i}(S_i) \leqslant \varepsilon \left(\sum_i \text{poids}_{G_i}(S_i) \right) \leqslant \varepsilon \left(k\frac{F}{k} + \sum_e c_e d_e \right) = 2\varepsilon F.$$

□

Théorème 20.7 *L'algorithme 20.4 est une $O(\log k)$-approximation pour le problème de la multicoupe minimum.*

Preuve : Il suffit de considérer à la suite les lemmes 20.5 et 20.6, et d'observer que la valeur F d'une multicoupe fractionnaire optimale est un minorant de la capacité d'une multicoupe minimum. □

Corollaire 20.8 *Dans un graphe non orienté avec k paires source-puits,*

$$\max_{\text{multiflot } F} |F| \quad \leqslant \min_{\text{multicoupe } C} |C| \quad \leqslant \quad O(\log k) \left(\max_{\text{mutliflot } F} |F| \right),$$

où $|F|$ désigne la valeur du multiflot F, et $|C|$ la capacité de la multicoupe C.

20.3 Une instance critique

Exemple 20.9 Nous allons construire une famille infinie de graphes pour lesquels le saut intégral du programme linéaire (20.2) est $\Omega(\log k)$, démontrant donc que notre analyse de l'algorithme 20.4 et le théorème du multiflot maximum et de la multicoupe minimum (corollaire 20.8) sont précis à un facteur constant près.

Cette construction utilise des expandeurs. Un *expandeur*[6] est un graphe $G = (V, E)$ où tous les sommets ont le même degré d, et où pour tout sous-ensemble non vide $S \subset V$,

$$|\delta(S)| > \min(|S|, |\overline{S}|),$$

où $\delta(S)$ désigne l'ensemble des arêtes de la coupe $(S, \overline{S})$, c'est-à-dire les arêtes qui ont une extrémité dans S et l'autre dans $\overline{S}$. Des arguments probabilistes standard démontrent que la quasi-totalité des graphes de degré constant $d = 3$, sont des expandeurs (voir section 20.6). Soit H un tel graphe à k sommets.

Construisons maintenant les paires source-puits dans H. Considérons les boules centrées en un sommet v quelconque. Le nombre de sommets à distance $\leqslant \alpha - 1$ du sommet v est inférieur à $1 + d + d^2 + \cdots + d^{\alpha-1} < d^\alpha$. Prenons $\alpha = \lfloor \log_d k/2 \rfloor$ pour garantir qu'au moins $k/2$ sommets sont à distance $\geqslant \alpha$ de v. L'ensemble des paires source-puits sera l'ensemble des paires de sommets à distance $\geqslant \alpha$ l'un de l'autre. Nous obtenons ainsi exactement $\Theta(k^2)$ paires de sommets source-puits (en considérant tous les choix de v possibles).

Nous donnons une capacité unitaire à toutes les arêtes de H. La capacité totale des arêtes de H est donc $O(k)$. Comme la distance entre toute source et son puits est $\Omega(\log k)$, tout chemin charriant une unité de flot utilise au moins $\Omega(\log k)$ unités de capacité. Ainsi, la valeur du multiflot maximum dans H est en $O(k/\log k)$. Nous allons démontrer que toute multicoupe minimum M de H a pour capacité $\Omega(k)$, d'où le saut intégral proclamé. Étudions les composantes connexes obtenues par le retrait de M dans H.

Fait 20.10 *Chaque composante connexe compte moins de $k/2$ sommets.*

Preuve : Supposons qu'une des composantes connexes ait strictement plus de $k/2$ sommets. Soit v un sommet arbitraire de cette composante. Par construction, le nombre de sommets à distance $\alpha - 1$ de v dans le graphe entier H est $< d^\alpha \leqslant k/2$. Ainsi, il existe un sommet u dans la composante tel que u et v soient à distance $\geqslant \alpha$ l'un de l'autre, et forment donc une paire source-puits : le retrait de M ne déconnecterait par toutes les paires, contradiction. □

D'après le fait 20.10, et puisque H est un expandeur, pour toute composante S, $|\delta(S)| \geqslant |S|$. Puisque chaque sommet de H appartient à des composantes, $\sum_S |\delta(S)| \geqslant k$, où la somme est prise sur toutes les composantes

[6] *Expander graph*, en anglais.

connexes. Comme chaque arête contribue au plus aux coupes de deux composantes, le nombre d'arêtes entre les composantes est $\Omega(k)$, d'où la minoration désirée sur la multicoupe minimum.

Reste à assurer que notre minoration du saut intégral est indépendante du nombre de sommets du graphe (k pour l'instant). Remarquons que remplacer une arête de H par un chemin d'arêtes de capacité unitaire ne modifie pas la valeur du multiflot maximum, ni celle de la multicoupe minimum. Nous pouvons donc construire, à partir de H, un graphe G à n sommets pour tout $n \geqslant k$, tel que le saut intégral du programme (20.2) sur le graphe G soit $\Omega(\log k)$. □

20.4 Quelques applications du problème de la multicoupe

Nous allons construire par réduction au problème de la multicoupe, une $O(\log n)$-approximation pour le problème suivant. Reportez-vous à l'exercice 20.7 pour d'autres applications.

Problème 20.11 (Élimination de clauses 2CNF≡)[7] Une *formule 2CNF≡* est un ensemble de clauses de la forme $(u \equiv v)$, où u et v sont des littéraux. Soit F une telle formule, munie de poids positifs et rationnels sur les clauses. Le problème est d'éliminer un sous-ensemble de clauses de F, de poids total minimum, tel que la formule obtenue soit satisfaisable.

À toute formule 2CNF≡ F sur n variables booléennes, nous associons un graphe $G(F)$ avec des capacités sur les arêtes : le graphe a $2n$ sommets, un par littéral. Deux arêtes pq et $\overline{p}\overline{q}$ sont créées pour chaque clause $(p \equiv q)$; la capacité de chacune de ces arêtes est le poids de la clause $(p \equiv q)$.

Remarquons que les deux clauses $(p \equiv q)$ et $(\overline{p} \equiv \overline{q})$ sont équivalentes. Nous pouvons supposer, sans perte de généralité, que F n'a pas de clauses équivalentes, car il suffit de cumuler leurs poids sur l'une et d'éliminer l'autre. Maintenant, à chaque clause correspondent deux arêtes distinctes de $G(F)$.

Lemme 20.12 *La formule F est satisfaisable ssi aucune composante connexe de $G(F)$ ne contient à la fois une variable et sa négation.*

Preuve : Pour toute arête pq de $G(F)$, les littéraux p et q doivent prendre la même valeur dans toute instanciation satisfaisant F. Ainsi, tous les littéraux d'une même composante connexe de $G(F)$ doivent prendre la même valeur. Par conséquent, si F est satisfaisable, aucune composante connexe de $G(F)$ ne contient un littéral et sa négation.

Réciproquement, remarquons que si les littéraux p et q appartiennent à la même composante connexe, alors il en est de même pour leurs négations.

[7] *2CNF≡ clause deletion*, en anglais.

Si aucune composante connexe ne contient une variable et sa négation, nous pouvons apparier les composantes connexes de telle sorte que dans chaque paire, l'une des composantes contient les littéraux niés de l'autre. Dans chaque paire, fixons les littéraux d'une composante à Vrai et ceux de l'autre à Faux. Nous obtenons une instanciation satisfaisant F. □

À chaque variable x_i, correspond une paire de sommets source-puits, x_i et $\overline{x}_i$, dans $G(F)$. Soient M, une multicoupe minimum de $G(F)$ pour ces n paires source-puits, et C, un ensemble de clauses de poids minimum dont l'élimination rend F satisfaisable. En général, M peut ne contenir que l'une des deux arêtes associées à une clause.

Lemme 20.13 $\text{poids}(C) \leqslant c(M) \leqslant 2 \cdot \text{poids}(C)$.

Preuve : Éliminons de F toutes les clauses associées aux arêtes de M. Soit F' la formule obtenue. Le poids des clauses éliminées est inférieur à $c(M)$. Puisque $G(F')$ ne contient aucune arête de M, aucune de ses composantes connexes ne contient une variable et sa négation. D'après le lemme 20.12, F' est donc satisfaisable, d'où la première inégalité.

Éliminons de $G(F)$ les deux arêtes associées à chacune des clauses de C. Toutes les paires source-puits sont déconnectées. Puisque la capacité des arêtes éliminées vaut $2\,\text{poids}(C)$, nous obtenons la seconde inégalité. □

Or, nous avons une $O(\log n)$-approximation pour la multicoupe minimum, donc :

Théorème 20.14 *Le problème 20.11 admet une $O(\log n)$-approximation.*

20.5 Exercices

20.1 Proposez un programme linéaire, équivalent au programme (20.1), qui définit une variable $f_{e,i}$ pour chaque arête e et chaque flot i, et qui utilise un nombre polynomial de variables. Exhibez son programme dual et démontrez qu'il est équivalent au programme (20.2). Mais, contrairement au programme (20.2), il n'a qu'un nombre polynomial de contraintes.

20.2 Soit $\boldsymbol{d}$ une solution optimale du programme (20.2). Démontrez que $\boldsymbol{d}$ est métrique (c'est-à-dire vérifie l'inégalité triangulaire).

20.3 Intuitivement, pour obtenir une bonne multicoupe, il faut sélectionner les arêtes qui sont des goulets d'étranglement pour le multiflot. De ce point de vue, D est un très bon point de départ : démontrez que D est précisément l'ensemble des arêtes saturées par *tous* les multiflots maximum.
Indication : Utilisez le théorème des écarts complémentaires.

20.4 Exhibez une instance démontrant que sélectionner tous les sommets de D peut donner une multicoupe à un facteur $\Omega(n)$ de l'optimum.

20.5 Étudions le processus d'expansion suivant. Notons $W(t)$ le poids à l'instant t. Donnons un poids initial $W(0) = W_0$ et supposons que le taux d'expansion est proportionnel au poids courant, c'est-à-dire

$$\mathrm{d}W(t) = \varepsilon W(t)\mathrm{d}t.$$

Déterminez la fonction $W(t)$. Puis, supposez $W_0 = F/k$ et $W(1/2) = F+F/k$. Que vaut ε ?
Indication : $W(t) = W_0 e^{\varepsilon t}$ et $\varepsilon = 2\ln(k+1)$.

20.6 Cet exercice explique le choix de poids(s_1), fixé à F/k. Supposons que nous le fixions à W_0. Clairement, ε est fonction de l'inverse de W_0 (voir lemme 20.3). Or le facteur d'approximation de l'algorithme vaut $\varepsilon \cdot (F + kW_0)$ (voir lemme 20.6). Quelle est la valeur de W_0 qui minimise le facteur d'approximation ?

20.7 Étudions le problème suivant lié à la conception de circuits VLSI.

Problème 20.15 (Bipartition par élimination d'arêtes)[8] Étant donné un graphe non orienté $G = (V, E)$ avec des poids sur les arêtes, trouver un ensemble d'arêtes de poids minimum, dont l'élimination de G donne un graphe biparti.

Exhibez une $O(\log n)$-approximation pour ce problème, en le réduisant au problème 20.11.

20.8 (Even, Naor, Schieber, et Rao [82]) Cet exercice construit une $O(\log^2 n)$-approximation pour le problème suivant.

Problème 20.16 (Arrangement linéaire de longueur minimum)[9] Étant donné un graphe non orienté $G = (V, E)$, trouver une numérotation des sommets de 1 à n, $h : V \to \{1, \ldots, n\}$, qui minimise

$$\sum_{uv \in E} |h(u) - h(v)|.$$

1. Démontrez que le programme suivant est une relaxation de ce problème. Ce programme linéaire associe à chaque arête $e \in E$ une variable d_e, vue comme un indicateur de distance. Pour toute instanciation des indicateurs de distance $\boldsymbol{d}$ sur les arêtes de G, notons $\mathrm{dist}_d(u, v)$ la longueur du

[8] *Graph bipartization by edge deletion*, en anglais.
[9] *Minimum length linear arrangement*, en anglais.

plus court chemin entre u et v dans G. Exhibez un oracle séparateur polynomial pour ce programme linéaire. En conclure qu'on peut le résoudre en temps polynomial.

$$\begin{aligned} \text{minimiser} \quad & \sum_{e \in E} d_e && (20.3) \\ & \sum_{u \in S} \mathrm{dist}_d(u, v) \geqslant \frac{1}{4}(|S|^2 - 1), && S \subseteq V, v \in S \\ & d_e \geqslant 0, && e \in E \end{aligned}$$

2. Soit $\boldsymbol{d}$ une solution optimale du programme (20.3). Montrez que pour tous $S \subseteq V$ et $v \in S$, il existe un sommet $u \in S$ tel que $\mathrm{dist}_d(u, v) \geqslant (|S|+1)/4$.
3. Pour tout $S \subseteq V$, notons poids(S) la somme des indicateurs de distance de toutes les arêtes dont les deux extrémités appartiennent à S, et $c(S, \overline{S})$ le nombre d'arêtes dans la coupe $(S, \overline{S})$. Proposez un processus d'expansion de régions, similaire à celui décrit section 20.2.1, qui construit une coupe $(S, \overline{S})$ de G, telle que poids$(S) \leqslant$ poids$(\overline{S})$ et telle que $c(S, \overline{S})$ soit en $O(\text{poids}(S)(\log n)/n)$.
4. Démontrez que la stratégie diviser-pour-régner, qui calcule récursivement une numérotation des sommets de S de 1 à $|S|$, et une numération de $\overline{S}$ de $|S| + 1$ à n, est une $O(\log^2 n)$-approximation.

   Indication : Supposez que la longueur de chaque arête de la coupe $(S, \overline{S})$ est $n - 1$, et écrivez la récurrence adéquate pour la fonction de coût de l'algorithme.

20.6 Notes

Le théorème 20.7 et son corollaire sont dus à Garg, Vazirani et Yannakakis [104]. Le problème 20.11 a été introduit par Klein, Rao, Agrawal et Ravi [181]. Se reporter à Pinsker [229] pour une preuve probabiliste de l'existence des expandeurs.

21 Coupe la moins dense

Ce chapitre présente une approximation pour le problème de la coupe la moins dense, mettant en œuvre une intéressante procédure d'arrondi fondée sur des plongements de métriques dans des espaces ℓ_1 ayant une faible distorsion. Comme annoncé au chapitre 20, nous en déduisons une double-inégalité de type théorème du flot maximum et de la coupe minimum pour le problème du multiflot sur demande. Nous en tirons des algorithmes d'approximation pour d'autres problèmes importants, tels que le calcul du taux de mélange d'une chaîne de Markov ou le calcul d'une coupe équilibrée.

21.1 Multiflot sur demande

Problème 21.1 (Multiflot sur demande)[1] Soient $G = (V, E)$ un graphe non orienté où chaque arête $e \in E$ a une capacité positive c_e, et un ensemble de paires de sommets distingués $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, où toutes les paires sont distinctes (les sommets de deux paires différentes peuvent être identiques). À chaque paire (s_i, t_i) est associé un flot différent, dont s_i est la source et t_i le puits. Une demande positive $\mathrm{dem}(i)$ est formulée pour chaque flot i. Le but est de maximiser le *débit*[2] f, tel que pour chaque flot i, $f \cdot \mathrm{dem}(i)$ unités du flot i soient routées simultanément, sous les contraintes de conservation de flot et de capacités des arêtes, c'est-à-dire : chaque flot doit se conserver en tout sommet différent de sa source et de son puits ; et la somme des flots traversant une arête, dans n'importe quelle direction, doit être inférieure à sa capacité. Nous noterons f^* le débit optimal.

Soit $(S, \overline{S})$ une coupe de G. Notons $c(S)$ la capacité des arêtes de cette coupe et $\mathrm{dem}(S)$ la demande totale traversant cette coupe, c'est-à-dire

$$\mathrm{dem}(S) = \sum_{i \,:\, |\{s_i, t_i\} \cap S| = 1} \mathrm{dem}(i).$$

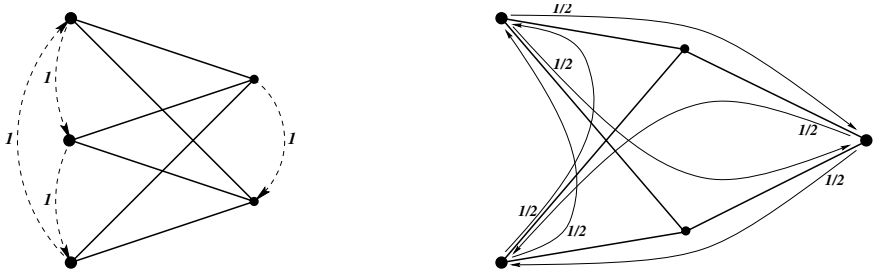
Clairement, le quotient de ces deux expressions majore le débit, c'est-à-dire $f^* \leqslant \frac{c(S)}{\mathrm{dem}(S)}$. Ceci nous conduit au problème suivant :

[1] *Demands multicommodity flow*, en anglais.
[2] *Throughput*, en anglais.

Problème 21.2 (Coupe la moins dense)[3] Considérons un graphe non orienté $G = (V, E)$ avec des capacités, des paires source-puits, et des demandes définies comme au problème 21.1. On appelle *densité*[4] d'une coupe $(S, \overline{S})$, la quantité $\frac{c(S)}{\text{dem}(S)}$. Le problème est trouver une coupe de densité, α^*, minimum.

Parmi toutes les coupes, α^* est le majorant le plus proche de f^*. Cette majoration est-elle très proche de f^* ? L'exemple 21.3 démontre que non. Cependant, la densité minimum n'est pas arbitrairement grande devant le débit maximum ; nous verrons que le quotient de ces deux quantités est un $O(\log k)$.

Exemple 21.3 Prenons le graphe biparti complet $K_{3,2}$ avec des capacités unitaires sur les arêtes et où une unité de flot est demandée d'un élément à l'autre de chaque paire de sommets non adjacents – soit quatre flots.



Il est facile de vérifier que la densité de la coupe la moins dense de $K_{3,2}$ vaut 1. Ce graphe est l'union de deux étoiles $K_{3,1}$ (dont les centres sont les sommets de la composante de droite), et comme dans l'exemple 18.2, il n'existe qu'une seule façon de transporter une unité de flot d'une source à un puits dans la composante de gauche. Celle-ci sature toutes les arêtes, et interdit donc de router le quatrième flot. Par conséquent, le débit est strictement inférieur à 1. □

21.2 Formulation par programmation linéaire

Voici un programme linéaire équivalent au problème du débit maximum. Notons $\mathcal{P}_i = \{q_j^i\}$ l'ensemble de tous les chemins entre s_i et t_i. Introduisons une variable f_j^i mesurant la quantité de flot i passant par le chemin q_j^i. Le premier jeu de contraintes garantit que les demandes sont satisfaites pour chaque flot (multipliées par f), et le second assure que les contraintes de capacité sur les arêtes sont vérifiées.

[3] *Sparsest cut*, en anglais.
[4] *Sparsity*, en anglais.

$$
\begin{array}{lll}
\text{maximiser} & f & \qquad (21.1)\\
\text{sous les contraintes} & \displaystyle\sum_j f_j^i \geqslant f \cdot \mathrm{dem}(i), & i = 1, \ldots, k\\
& \displaystyle\sum_{q_j^i \,:\, e \in q_j^i} f_j^i \leqslant c_e, & e \in E\\
& f \geqslant 0 & \\
& f_j^i \geqslant 0 &
\end{array}
$$

Nous appellerons *graphe des demandes*, le graphe H ayant pour sommets $V_H = \{s_i, t_i \,:\, 1 \leqslant i \leqslant k\}$ et pour arêtes $E_H = \{(s_i, t_i) \,:\, 1 \leqslant i \leqslant k\}$; posons $\mathrm{dem}(e) = \mathrm{dem}(i)$, pour chaque arête $e = (s_i, t_i)$ de H. Nous allons démontrer que le programme dual de (21.1) définit un espace métrique $(V, \boldsymbol{d})$ qui vérifie la propriété suivante :

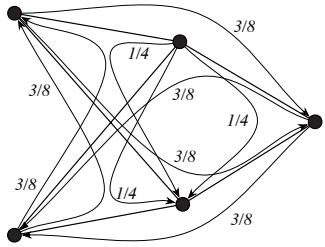
Théorème 21.4 *Soit f^* le débit optimal. Alors,*

$$f^* \;=\; \min_{\boldsymbol{d}\ \textit{métrique}} \frac{\sum_{e \in G} c_e d_e}{\sum_{e \in H} \mathrm{dem}(e) d_e}.$$

Preuve : Notons l_i et d_e les variables duales associées respectivement aux premier et second jeux d'inégalités du programme (21.1). Nous interprétons (d_e) comme des indicateurs de distance sur les arêtes de G. Le premier jeu d'inégalités garantit que pour chaque flot i, l_i est inférieur à la longueur (définie par les indicateurs de distance) de tout chemin entre s_i et t_i.

$$
\begin{array}{lll}
\text{minimiser} & \displaystyle\sum_{e \in E} c_e d_e & \qquad (21.2)\\
\text{sous les contraintes} & \displaystyle\sum_{e \in q_j^i} d_e \geqslant l_i, & q_j^i \in \mathcal{P}_i, i = 1, \ldots, k\\
& \displaystyle\sum_{i=1}^{k} l_i \,\mathrm{dem}(i) \geqslant 1 & \\
& d_e \geqslant 0, & e \in E\\
& l_i \geqslant 0, & i = 1, \ldots, k
\end{array}
$$

Exemple 21.5 Pour l'instance de l'exemple 21.3, le débit optimal est $f^* = 3/4$; ceci revient à router les quatre flots ainsi :



Une solution duale optimale est : $d_e = 1/8$, pour chaque arête e, et $l_i = 1/4$, pour chaque flot i — nous encourageons vivement le lecteur à vérifier la faisabilité et l'optimalité de ces solutions. □

Fait 21.6 *Il existe une instanciation des indicateurs de distance* $\boldsymbol{d}$ *pour le programme dual (21.2) qui définit une métrique sur* V*. De plus, pour chaque flot* i*,* $l_i = d(s_i, t_i)$ *et la seconde inégalité est une égalité, c'est-à-dire* $\sum_i d(s_i, t_i)\,\mathrm{dem}(i) = 1$*.*

Preuve : S'il existe trois points u, v et w tels que $d_{uw} > d_{uv} + d_{vw}$, alors nous fixons la valeur de d_{uw} à $(d_{uv} + d_{vw})$. Cela ne raccourcit pas les plus courts chemins entre les paires (s_i, t_i), et la solution est donc toujours réalisable. De plus, la valeur objectif reste inchangée. Itérer ce processus conduit donc à une métrique sur V.

La longueur du plus court chemin entre s_i et t_i est maintenant donnée par l'indicateur de distance $d(s_i, t_i)$. Poser $l_i = d(s_i, t_i)$ ne modifie ni la faisabilité, ni la valeur objectif de la solution. Enfin, si la seconde inégalité est stricte, on peut réduire tous les indicateurs de distance proportionnellement, sans violer la faisabilité, ce qui contredit l'optimalité de $\boldsymbol{d}$. □

D'après le fait 21.6, le programme dual définit une métrique $(V, \boldsymbol{d})$ qui minimise :

$$\frac{\sum_{e \in G} c_e d_e}{\sum_{e \in H} \mathrm{dem}(e) d_e}.$$

D'après le théorème de dualité en programmation linéaire, c'est aussi le débit optimal. Nous avons donc démontré le théorème 21.4. □

21.3 Métriques, empaquetage de coupes et plongements ℓ_1

Dans la section 21.3.1, nous définissons la notion d'empaquetage de coupes pour une métrique et démontrons que trouver une bonne approximation pour

la coupe la moins dense d'un graphe G revient à trouver un « bon » empaquetage de coupes pour la métrique définie au théorème 21.4. La section 21.3.2 réduit ce dernier problème à la recherche d'un « bon » plongement ℓ_1 pour la métrique.[5] Enfin, la section 21.4 explique comment construire ce plongement.

21.3.1 Empaquetages de coupes pour une métrique

Une métrique $(V, \boldsymbol{d})$ définit des longueurs pour les arêtes d'un graphe complet sur V. Notons E_n l'ensemble des arêtes du graphe complet à n sommets. Soit $y : 2^V \to \mathbf{R}^+$ une fonction positive sur les sous-ensembles de V. Notons y_S la valeur de y sur le sous-ensemble S. Nous dirons que l'arête e est couverte par y_S si e appartient à la coupe $(S, \overline{S})$. Le *niveau de couverture total* de l'arête e vaut $\sum_{S\,:\,e\in\delta(S)} y_S$. La fonction y est un *empaquetage de coupes*[6] pour la métrique $(V, \boldsymbol{d})$ si le niveau de couverture total de chaque arête est inférieur à sa longueur, c'est-à-dire si pour toute arête $e \in E_n$, $\sum_{S\,:\,e\in\delta(S)} y_S \leqslant d_e$. Si cette inégalité est une égalité pour chaque arête $e \in E_n$, alors l'empaquetage de coupes y est dit *exact*. On emploie le terme « empaquetage de coupes » car cela revient à interpréter y comme la sélection de $y_S + y(\overline{S})$ tantièmes de chaque coupe $(S, \overline{S})$.

Comme nous le verrons plus bas, il n'existe pas en général d'empaquetage exact pour une métrique $(V, \boldsymbol{d})$ donnée. Nous relâchons cette notion en imposant que les arêtes soient couvertes à un taux minimum. Pour $\beta \geqslant 1$, y est un empaquetage de coupes à β près,[7] si le niveau de couverture de chaque arête est supérieur à sa longueur divisée par β, c'est-à-dire si pour toute arête $e \in E_n$, $d_e/\beta \leqslant \sum_{S\,:\,e\in\delta(S)} y_S \leqslant d_e$. Plus β est proche de 1, meilleur est l'empaquetage de coupes. L'importance de la recherche d'un bon empaquetage de coupes pour $(V, \boldsymbol{d})$ apparaît au théorème suivant.

Théorème 21.7 *Soient $(V, \boldsymbol{d})$ la métrique définie par le théorème 21.4, et y un empaquetage de coupes à β près pour $(V, \boldsymbol{d})$. Soit $(S', \overline{S'})$ la coupe la moins dense parmi toutes les coupes S telles que $y_S \neq 0$. Alors, la densité de cette coupe est inférieure à $\beta \cdot f^*$.*

Preuve : Soit y un empaquetage de coupes à β près pour la métrique $(V, \boldsymbol{d})$. Alors,

$$\begin{aligned} f^* &= \frac{\sum_{e\in G} c_e d_e}{\sum_{e\in H} \operatorname{dem}(e) d_e} \geqslant \frac{\sum_{e\in G} c_e \sum_{S\,:\,e\in\delta(S)} y_S}{\sum_{e\in H} \operatorname{dem}(e) \sum_{S\,:\,e\in\delta(S)} \beta y_S} \\ &= \frac{\sum_S y_S c(S)}{\beta \sum_S y_S \operatorname{dem}(S)} \\ &\geqslant \frac{1}{\beta} \cdot \left(\frac{c(S')}{\operatorname{dem}(S')} \right). \end{aligned}$$

[5] *ℓ_1-embedding*, en anglais.
[6] *Cut packing*, en anglais.
[7] *β-Approximate cut packing*, en anglais.

La première inégalité utilise la majoration et la minoration sur le niveau de couverture d'une arête – la première pour le numérateur et la seconde pour le dénominateur. L'égalité qui suit est une simple réorganisation de la somme. La dernière inégalité est l'application du résultat bien connu énoncé ci-après. □

Proposition 21.8 *Pour tous réels positifs $a_1, \ldots, a_n$ et tous réels strictement positifs $b_1, \ldots, b_n$ et $\alpha_1, \ldots, \alpha_n$,*

$$\frac{\sum_i \alpha_i a_i}{\sum_i \alpha_i b_i} \geqslant \min_i \frac{a_i}{b_i}.$$

De plus, cette inégalité est une égalité ssi les n quotients a_i/b_i sont tous égaux.

Corollaire 21.9 *S'il existe un empaquetage exact pour la métrique $(V, \boldsymbol{d})$, alors toute coupe $(S, \overline{S})$ telle que $y_S \neq 0$ a pour densité f^*, c'est-à-dire est une coupe la moins dense de G.*

Preuve : D'après le théorème 21.7, une coupe de densité minimum telle que $y_S \neq 0$ a une densité inférieure à f^* (car $\beta = 1$). Comme la densité de toute coupe majore f^*, la densité de cette coupe vaut f^* et c'est une coupe la moins dense de G. Toutes les inégalités de la preuve du théorème 21.7 sont donc des égalités. La seconde partie de la proposition 21.8 implique par conséquent que la densité de toutes les coupes $(S, \overline{S})$ telles que $y_S \neq 0$, vaut f^*. □

La densité de la coupe la moins dense de l'instance de l'exemple 21.3 est strictement supérieure à f^*. D'après le corollaire 21.9, la métrique optimale de cette instance n'admet donc pas d'empaquetage de coupes exact. Nous allons démontrer, en utilisant la notion de plongement ℓ_1 de métriques, que toute métrique admet cependant un empaquetage de coupes à $O(\log n)$ près.

21.3.2 Plongement ℓ_1 d'une métrique

Une *norme* sur l'espace vectoriel $\mathbf{R}^m$ est une fonction $\|\cdot\| : \mathbf{R}^m \to \mathbf{R}^+$, telle que pour tous $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^m$ et $\lambda \in \mathbf{R}$:

- $\|\boldsymbol{x}\| = 0$ ssi $\boldsymbol{x} = 0$,
- $\|\lambda \boldsymbol{x}\| = |\lambda| \cdot \|\boldsymbol{x}\|$,
- $\|\boldsymbol{x} + \boldsymbol{y}\| \leqslant \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$.

Pour $p \geqslant 1$, la *norme* ℓ_p est définie par

$$\|\boldsymbol{x}\|_p = \left(\sum_{1 \leqslant k \leqslant m} |x_k|^p \right)^{\frac{1}{p}}.$$

On note $\boldsymbol{d}_{\ell_p}$ la métrique ℓ_p associée, définie pour tous $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^m$ par :

$$\boldsymbol{d}_{\ell_p}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_p.$$

Dans cette section, nous n'étudierons que la norme ℓ_1.

Pour un m donné, considérons un plongement $\sigma : V \to \mathbf{R}^m$ des sommets du graphe dans $\mathbf{R}^m$. Nous dirons que $\|\sigma(u) - \sigma(v)\|_1$ est la *longueur ℓ_1 de l'arête uv selon σ*. σ est appelé un *plongement ℓ_1 isométrique* pour la métrique $(V, \boldsymbol{d})$ s'il conserve les longueurs des arêtes pour la norme ℓ_1, c'est-à-dire si

$$\forall u, v \in V, \quad d(u, v) = \|\sigma(u) - \sigma(v)\|_1.$$

Nous verrons ci-dessous qu'il n'existe pas en général de plongement ℓ_1 isométrique pour les métriques calculées par le programme dual. Nous allons donc relâcher cette notion, en imposant que chaque arête ne soit *étirée* ou *contractée* qu'au plus dans une certaine proportion par le plongement. Pour $\beta \geqslant 1$, σ est appelé *plongement ℓ_1 de distorsion β* pour une métrique $(V, \boldsymbol{d})$ si

$$\forall u, v \in V, \quad \frac{1}{\beta}\, d(u, v) \leqslant \|\sigma(u) - \sigma(v)\|_1 \leqslant d(u, v).$$

Par la suite, nous verrons que la recherche d'une bonne approximation d'un empaquetage de coupes pour une métrique donnée est intimement liée à la recherche d'un plongement ℓ_1 de faible distorsion pour cette métrique.

Lemme 21.10 *Soit $\sigma : V \to \mathbf{R}^m$ un plongement. Il existe un empaquetage de coupes $y : 2^V \to \mathbf{R}^+$ tel que le niveau de couverture de chaque arête par y est égal à sa longueur ℓ_1 selon σ. De plus, le nombre de y_S non nuls est inférieur à $m(n-1)$.*

Preuve : Commençons par le cas $m = 1$. Notons $u_1 \leqslant u_2 \leqslant \cdots \leqslant u_n$ les images des sommets de $V = \{v_1, \ldots, v_n\}$ par σ. Pour chaque $i, 1 \leqslant i \leqslant n-1$, posons $y_{\{v_1,\ldots,v_i\}} = u_{i+1} - u_i$. Clairement, cet empaquetage de coupes vérifie les propriétés requises.

Pour le cas général, remarquons que la norme ℓ_1 est additive[8]. Nous pouvons donc définir un empaquetage de coupes pour chaque dimension indépendamment, la somme de ces empaquetages vérifiera les propriétés voulues. □

Lemme 21.11 *Soient $y : 2^V \to \mathbf{R}^+$ un empaquetage de coupes et m le nombre d'entrées y_S non nulles. Il existe un plongement $\sigma : V \to \mathbf{R}^m$, telle*

[8] c'est-à-dire la norme ℓ_1 d'un vecteur est la somme des normes suivant chaque dimension : $\|x\|_1 = \sum_i \|x_i\|_1$.

que la longueur ℓ_1 de toute arête selon σ, est égale à son niveau de couverture par y.

Preuve : À chaque sous-ensemble $S \subseteq V$ tel que $y_S \neq 0$, nous associons une des coordonnées. Cette coordonnée est fixée à 0 pour les sommets de S et à y_S pour les sommets de $\overline{S}$. Ainsi, la contribution de cette coordonnée à la longueur ℓ_1 de chaque arête est égale au niveau de couverture y_S de l'arête par S. Ce plongement satisfait donc bien les conditions requises. □

Les lemmes 21.10 et 21.11 donnent :

Théorème 21.12 *Il existe un plongement ℓ_1 de distorsion β pour la métrique $(V, \boldsymbol{d})$ ssi il existe un empaquetage de coupes à β près pour cette métrique. De plus, le nombre de coupes non nulles et la dimension du plongement ℓ_1 sont en relation polynomiale.*

Corollaire 21.13 *Il existe un plongement ℓ_1 isométrique pour une métrique $(V, \boldsymbol{d})$ ssi il existe un empaquetage de coupes exact pour cette métrique.*

Nous avons vu que la métrique obtenue sur l'instance de l'exemple 21.3 n'admet pas d'empaquetage de coupes exact. Par conséquent, il n'admet pas de plongement ℓ_1 isométrique. Cependant, nous allons démontrer que toute métrique admet un plongement ℓ_1 de distorsion $O(\log n)$; c'est le cœur de l'algorithme d'approximation pour le problème de la coupe la moins dense.

21.4 Plongement ℓ_1 de faible distorsion d'une métrique

Commençons par étudier le plongement ℓ_1 unidimensionnel suivant d'une métrique $(V, \boldsymbol{d})$: fixer un ensemble $S \subseteq V$, et définir la coordonnée du sommet v par $\sigma(v) = \min_{s \in S} d(s, v)$, c'est-à-dire la longueur de l'arête la plus courte de v à S. Ce plongement n'étire aucune arête :

Lemme 21.14 *Pour le plongement unidimensionnel σ décrit ci-dessus :*

$$\forall u, v \in V, \quad |\sigma(u) - \sigma(v)| \leqslant d(u, v).$$

Preuve : Soient s_1 et s_2 deux sommets de S à distance minimum de u et v, respectivement. Quitte à échanger u et v, supposons $d(s_1, u) \leqslant d(s_2, v)$. Alors, $|\sigma(u) - \sigma(v)| = d(s_2, v) - d(s_1, u) \leqslant d(s_1, v) - d(s_1, u) \leqslant d(u, v)$. La dernière inégalité est une conséquence de l'inégalité triangulaire. □

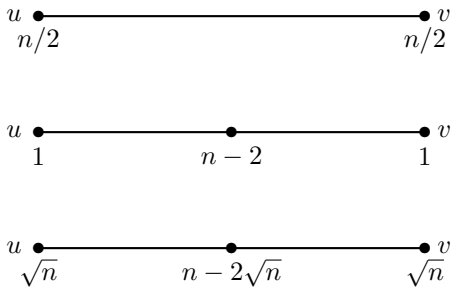
Plus généralement, étudions le plongement m-dimensionnel suivant : fixer m sous-ensembles $S_1, \ldots, S_m$ de V, et définir la i-ième coordonnée du sommet v par $\sigma_i(v) = \min_{s \in S_i} d(s, v)/m$ (notez la division par m). Par additivité de la norme ℓ_1, et d'après le lemme 21.14, ce plongement n'étire aucune arête non plus.

21.4.1 Garantir qu'une arête donnée ne soit pas trop contractée

Il faut choisir les ensembles $S_1, \ldots, S_m$ de sorte qu'aucune arête ne soit contractée par un facteur supérieur à $\Theta(\log n)$. Une façon naturelle de procéder est d'utiliser la randomisation. Commençons par garantir qu'une arête donnée ne soit pas trop contractée. Pour cela, définissons l'*espérance de la contribution d'un ensemble* S_i à la longueur ℓ_1 de l'arête uv comme $\mathbf{E}[|\sigma_i(u) - \sigma_i(v)|]$.

Pour simplifier, supposons que n est une puissance de 2 ; posons $n = 2^l$. Pour $2 \leqslant i \leqslant l+1$, l'ensemble S_i est construit en sélectionnant chaque sommet de V avec probabilité $1/2^i$ indépendamment. Le plongement défini par ces ensembles fonctionne pour une arête donnée uv avec une forte probabilité. La preuve de ce fait nécessite de comptabiliser astucieusement l'espérance des contributions de chaque ensemble. Suivant les métriques, les ensembles ayant une grande contribution changent. Commençons par une série d'exemples qui nous permettrons de mieux appréhender la preuve.

Exemple 21.15 Considérons les trois métriques suivantes où $d(u, v) = 1$, et où les n sommets sont placés comme indiqué sur les figures ci-dessous.



Pour chacune de ces métriques, il existe un ensemble dont l'espérance de la contribution est $\Omega(d(u, v)/l)$. Pour la première métrique, cet ensemble est S_l, puisqu'il choisit un singleton avec probabilité constante. Pour la seconde, cet ensemble est S_2, car il contient un seul des deux sommets u et v avec probabilité constante. Et pour la troisième, cet ensemble est $S_{\lceil l/2 \rceil}$, puisqu'avec probabilité constante, il contient un seul des deux ensembles de $2/\sqrt{n}$ sommets placés sur u ou v. □

Le lemme suivant présente le mécanisme de minoration de l'espérance de la contribution de chaque ensemble S_i. Pour tout sommet x et tout réel $r \geqslant 0$, notons $B(x, r)$ la *boule* de rayon r autour de x, c'est-à-dire $B(x, r) = \{s \in V : d(x, s) \leqslant r\}$.

Lemme 21.16 *S'il existe $r_1 \geqslant r_2 \geqslant 0$, et une constante c, tels que*

$$\mathbf{Pr}[(S_i \cap B(u, r_1) = \varnothing) \text{ et } (S_i \cap B(v, r_2) \neq \varnothing)] \geqslant c,$$

alors l'espérance de la contribution de S_i *est* $\geqslant c(r_1 - r_2)/l$.

Preuve : Si l'événement décrit se produit, alors $d(u, S_i) > r_1$ et $d(v, S_i) \leqslant r_2$. Dans ce cas, $\sigma_i(u) > r_1/l$ et $\sigma_i(v) \leqslant r_2/l$. Et donc, $|\sigma_i(u) - \sigma_i(v)| > (r_1 - r_2)/l$, CQFD. □

Il reste à définir des valeurs adéquates pour les rayons r_1 et r_2 pour chaque ensemble S_i afin d'appliquer le lemme 21.16. Utilisons le lemme probabiliste élémentaire suivant :

Lemme 21.17 *Pour* $1 \leqslant t \leqslant l-1$, *soient* A *et* B *deux sous-ensembles disjoints de* V, *tels que* $|A| < 2^t$ *et* $|B| \geqslant 2^{t-1}$. *Si* S *est un ensemble construit en sélectionnant chaque sommet de* V *indépendamment avec probabilité* $p = 1/(2^{t+1})$, *alors :*

$$\mathbf{Pr}[(S \cap A = \varnothing) \text{ et } (S \cap B \neq \varnothing)] \geqslant (1 - e^{-1/4})/2.$$

Preuve :

$$\mathbf{Pr}[S \cap A = \varnothing] = (1-p)^{|A|} \geqslant (1 - p|A|) \geqslant \frac{1}{2},$$

où la première inégalité s'obtient en ne conservant que les deux premiers termes du développement en série entière.

$$\mathbf{Pr}[S \cap B = \varnothing] = (1-p)^{|B|} \leqslant e^{-p|B|} \leqslant e^{-1/4},$$

en utilisant que $1 - x \leqslant e^{-x}$. Ainsi,

$$\mathbf{Pr}[S \cap B \neq \varnothing] = 1 - (1-p)^{|B|} \geqslant 1 - e^{-1/4}.$$

Enfin, comme A et B sont disjoints, les événements $[S \cap A = \varnothing]$ et $[S \cap B \neq \varnothing]$ sont indépendants. CQFD. □

Posons $c = (1 - e^{-1/4})/2$. Pour $0 \leqslant t \leqslant l$, posons $\rho_t = \min\{\rho \geqslant 0 : |B(u,\rho)| \geqslant 2^t \text{ et } |B(v,\rho)| \geqslant 2^t\}$, c'est-à-dire ρ_t est le plus petit rayon tel que les boules de rayon ρ_t centrées en u et en v contiennent plus de 2^t sommets. Clairement, $\rho_0 = 0$ et $\rho_l \geqslant d(u,v)$. Soit $\hat{t} = \max\{t : \rho_t < d(u,v)/2\}$; clairement, $\hat{t} \leqslant l-1$. Enfin, pour tout sommet x et réel positif r, notons $\mathring{B}(x,r)$ la *boule ouverte* de rayon r centrée en x, c'est-à-dire $\mathring{B}(x,r) = \{s \in V : d(x,s) < r\}$.

Lemme 21.18 *Pour* $1 \leqslant t \leqslant \hat{t}$, *l'espérance de la contribution de* S_{t+1} *est supérieure à* $c \cdot \frac{\rho_t - \rho_{t-1}}{l}$, *et pour* $t = \hat{t} + 1$, *l'espérance de la contribution de* S_{t+1} *est supérieure à* $\frac{c}{l} \cdot \left(\frac{d(u,v)}{2} - \rho_{t-1}\right)$.

Preuve : Commençons par considérer $1 \leqslant t \leqslant \hat{t}$. Par définition de ρ_t, au moins l'une des deux boules ouvertes de rayon ρ_t centrées en u ou v contient strictement moins de 2^t sommets. Quitte à échanger u et v, supposons que ce soit celle qui est centrée en u, c'est-à-dire $|\mathring{B}(u, \rho_t)| < 2^t$. De nouveau, par définition, $|B(v, \rho_{t-1})| \geqslant 2^{t-1}$. Puisque $\rho_{t-1} \leqslant \rho_t < d(u,v)/2$, les deux boules $\mathring{B}(u, \rho_t)$ et $B(v, \rho_{t-1})$ sont disjointes. Ainsi, d'après le lemme 21.17, la probabilité que S_{t+1} n'intersecte pas la première boule et intersecte la seconde, est supérieure à c. Du lemme 21.16, nous déduisons donc le premier point.

Ensuite, prenons $t = \hat{t} + 1$. Par définition de $\hat{t}$, au moins l'une des deux boules ouvertes de rayon $d(u,v)/2$ centrée en u ou v, contient strictement moins de 2^t sommets. Quitte à échanger u et v, supposons que ce soit celle centrée en u, c'est-à-dire $|\mathring{B}(u, d(u,v)/2)| < 2^t$. Clairement, $|B(v, \rho_{t-1})| \geqslant 2^{t-1}$. Puisque $\rho_{t-1} < d(u,v)/2$, les deux boules $\mathring{B}(u, d(u,v)/2)$ et $B(v, \rho_{t-1})$ sont disjointes. Il suffit de reprendre les arguments précédents pour conclure. □

Lemme 21.19 *L'espérance de la contribution totale des ensembles $S_2, \ldots, S_{l+1}$ est supérieure à $\frac{c}{2} \cdot \frac{d(u,v)}{l}$.*

Preuve : D'après le lemme 21.18, l'espérance de la contribution des ensembles $S_2, \ldots, S_{l+1}$ est minorée par la somme télescopique suivante :

$$\frac{c}{l} \cdot \left((\rho_1 - \rho_0) + (\rho_2 - \rho_1) + \ldots + \left(\frac{d(u,v)}{2} - \rho_{\hat{t}} \right) \right) = \frac{c}{2} \cdot \frac{d(u,v)}{l}.$$

□

Lemme 21.20

$$\mathbf{Pr}\left[\text{contribution totale des ensembles est} \geqslant \frac{c\, d(u,v)}{4l} \right] \geqslant \frac{c/2}{2 - c/2}.$$

Preuve : Notons p la probabilité à minorer. Clairement, la contribution totale des ensembles $S_2, \ldots, S_{l+1}$ à la longueur ℓ_1 de l'arête uv est supérieure à $d(u,v)/l$. Ainsi, d'après le lemme 21.19,

$$p \cdot \frac{d(u,v)}{l} + (1-p) \cdot \frac{c\, d(u,v)}{4l} \geqslant \frac{c\, d(u,v)}{2l}.$$

Par conséquent, $p \geqslant \frac{c/2}{2 - c/2}$. □

21.4.2 Garantir qu'aucune arête ne soit trop contractée

Pour une arête uv donnée, avec une probabilité constante, le plongement précédent ne contracte pas trop cette arête. Afin d'assurer qu'aucune arête ne soit trop contractée, ensemble, nous allons « doper » cette probabilité. La clé est de répéter le processus entièrement plusieurs fois et indépendamment, puis d'utiliser les bornes de Chernoff pour majorer la probabilité d'échec. Nous utilisons la formulation suivante de la borne de Chernoff : soient $X_1, \ldots, X_n$ des variables zéro-un de Bernoulli indépendantes telles que $\mathbf{Pr}[X_i = 1] = p$, avec $0 < p < 1$, et posons $X = \sum_{i=1}^{n} X_i$ (clairement, $\mathbf{E}[X] = np$) ; alors, pour $0 < \delta \leqslant 1$,

$$\mathbf{Pr}[X < (1-\delta)\, n\, p] < \exp(-\delta^2 n\, p/2).$$

Sélectionnons indépendamment $N = \theta(\log n)$ fois chacun, des ensembles $S_2, \ldots, S_{l+1}$ suivant les lois de probabilités définies précédemment. Notons les ensembles obtenus S_i^j, avec $2 \leqslant i \leqslant l+1$ et $1 \leqslant j \leqslant N$. Considérons le plongement σ de la métrique $(V, \boldsymbol{d})$, dans un espace de dimension $N \cdot l = O(\log^2 n)$, associé à ces $N \cdot l$ ensembles. Et démontrons que σ est un plongement ℓ_1 de distorsion $O(\log n)$ pour la métrique $(V, \boldsymbol{d})$.

Lemme 21.21 *Si $N = \Omega(\log n)$, pour tout $u, v \in V$:*

$$\mathbf{Pr}\left[\|\sigma(u) - \sigma(v)\|_1 \geqslant \frac{p\, c\, d(u,v)}{8l}\right] \geqslant 1 - \frac{1}{2n^2},$$

avec $p = c/(4-c)$.

Preuve : Pour tout j, $1 \leqslant j \leqslant N$, voyons le processus de construction des ensembles $S_2^j, \ldots, S_{l+1}^j$ comme une unique variable zéro-un de Bernoulli tirée N fois indépendamment. Chaque tirage est un succès si la contribution totale des ensembles est $\geqslant (c\, d(u,v))/4l$. D'après, le lemme 21.20, la probabilité de succès est supérieure à p. D'après la borne de Chernoff avec $\delta = 1/2$, la probabilité que moins de $N\, p/2$ tirages soient des succès est inférieure à $\exp(-N\, p/8)$, lui-même majoré par $1/2n^2$ pour $N = \Omega(\log n)$. Comme la longueur ℓ_1 de l'arête uv est supérieure à $p\, c\, d(u,v)/8l = d(u,v)/O(\log n)$ si au moins $N\, p/2$ tirages sont des succès le lemme s'ensuit. □

En sommant les probabilités d'échec sur les $n(n-1)/2$ arêtes, nous obtenons :

Théorème 21.22 *Le plongement de dimension $N\, l = O(\log^2 n)$ donné ci-dessus est un plongement ℓ_1 de distorsion $O(\log n)$ pour la métrique $(V, \boldsymbol{d})$, avec probabilité supérieure à $1/2$.*

21.5 Algorithme par arrondi

Nous laissons au lecteur le soin de vérifier que le fait 21.6 et les théorèmes 21.7, 21.12, et 21.22 conduisent à une $O(\log n)$-approximation pour le problème de la coupe la moins dense. Dans cette section, nous construisons une $O(\log k)$-approximation, où k est le nombre de paires source-puits spécifiées.

Pour ce but, remarquons que le théorème 21.7 est toujours vrai avec les contraintes plus faibles suivantes sur l'empaquetage de coupes approché : aucune arête ne doit déborder, et toutes les arêtes du graphe des demandes sont totalement couvertes à un facteur β près (le niveau de remplissage des autres arêtes peut être quelconque). De même que précédemment, un tel empaquetage de coupes s'obtient en construisant un plongement ℓ_1 qui ne contracte pas trop les arêtes du graphe des demandes, sans contraintes sur les autres. Puisqu'il y a seulement $O(k^2)$ telles arêtes, nous pouvons assurer que ces arêtes sont contractées d'un facteur $O(\log k)$, permettant ainsi l'amélioration visée du facteur d'approximation.

Notons $V' \subseteq V$ l'ensemble des sources et des puits, $|V'| \leqslant 2k$. Pour simplifier, supposons que $|V'|$ est une puissance de 2 et posons $|V'| = 2^l$. Les ensembles $S_2, \ldots, S_{l+1}$ sont construits dans V', et il est facile d'adapter la preuve du lemme 21.21 pour montrer que $N = O(\log k)$ suffit pour assurer que les $O(k^2)$ arêtes du graphe des demandes soient contractées d'un facteur $O(\log k)$. Voici l'algorithme complet :

Algorithme 21.23 (Coupe la moins dense)

1. Résoudre le programme dual (21.2) et obtenir la métrique $(V, \boldsymbol{d})$.
2. Construire les ensembles $S_i^j,\ 2 \leqslant i \leqslant l+1,\ 1 \leqslant j \leqslant N$, où S_i^j est construit en sélectionnant indépendamment chaque sommet de V' avec probabilité $1/2^i$.
3. Construire le plongement ℓ_1 de $(V, \boldsymbol{d})$ dans un espace de dimension $O(\log^2 k)$, associé à ces ensembles.
4. Construire l'empaquetage de coupes approché pour $(V, \boldsymbol{d})$ associé au plongement ℓ_1.
5. Renvoyer la coupe la moins dense correspondant à cet empaquetage de coupes.

Théorème 21.24 *L'algorithme 21.23 est une $O(\log k)$-approximation pour le problème de la coupe la moins dense.*

Corollaire 21.25 *Pour le multiflot sur demande à k paires source-puits,*

$$\frac{1}{O(\log k)} \left(\min_{S \subset V} \frac{c(S)}{\mathrm{dem}(S)} \right) \leqslant \max_{\textit{débit } f} f \leqslant \min_{S \subset V} \frac{c(S)}{\mathrm{dem}(S)}.$$

21.6 Applications

Cette section présente différentes applications du problème de la coupe la moins dense.

21.6.1 Capacité d'expansion par arête

Les expandeurs ont de nombreuses applications, voir l'exemple 20.9. Nous allons construire une $O(\log n)$-approximation pour déterminer l'expansion par arête d'un graphe :

Problème 21.26 (Expansion par arête)[9] Étant donné un graphe non orienté $G = (V, E)$, l'*expansion par arête d'un sous-ensemble* $S \subset V$ avec $|S| \leqslant n/2$, est définie par $|\delta(S)|/|S|$. Le problème est de trouver un ensemble d'expansion par arête minimum.

Prenons le cas particulier du multiflot avec $n(n-1)/2$ flots distincts, un pour chaque paire de sommets. Ce problème est appelé le *multiflot uniforme*. Pour ce problème, la densité de n'importe quelle coupe $(S, \overline{S})$ vaut

$$\frac{c(S)}{|S| \cdot |\overline{S}|}.$$

Soit $(S, \overline{S})$, avec $|S| \leqslant |\overline{S}|$, la coupe construite par l'algorithme 21.23 sur le graphe G avec des demandes uniformes. Remarquez que $|\overline{S}|$ est connu à un facteur 2 près, car $n/2 \leqslant |\overline{S}| \leqslant n$. Ainsi, l'expansion de S est à un facteur $O(\log n)$ de l'expansion minimum d'un sous-ensemble de sommets de G. Clairement, la généralisation de ce problème au cas d'arêtes de poids arbitraires admet également une $O(\log n)$-approximation.

21.6.2 Conductance

La conductance d'une chaîne de Markov caractérise son taux de mélange, c'est-à-dire le nombre de pas nécessaires pour garantir que la distribution de probabilité sur les états est suffisamment proche de la distribution stationnaire. Notons P la matrice de transition d'une chaîne de Markov à temps discret sur un espace fini d'états X, et π sa distribution stationnaire. Nous supposerons que cette chaîne est apériodique, connexe et réversible, c'est-à-dire pour le dernier point, que

$$\pi(x)P(x, y) = \pi(y)P(y, x), \quad \forall x, y \in X.$$

Construisons le graphe non orienté $G = (X, E)$ ayant pour sommets les états X et tel que $xy \in E$ ssi $\pi(x)P(x, y) \neq 0$. Chaque arête xy a pour poids $w(xy) = \pi(x)P(x, y)$. La *conductance* Φ de la chaîne est définie par

[9] *Edge expansion*, en anglais.

$$\Phi = \min_{S \subset X,\ 0<\pi(S)\leqslant 1/2} \frac{w(S,\overline{S})}{\pi(S)},$$

où $w(S,\overline{S})$ est la somme des poids des arêtes de la coupe $(S,\overline{S})$. Pour tout ensemble S, le numérateur du quotient ci-dessus, est la probabilité de passer de l'ensemble S à $\overline{S}$ en un pas, depuis l'état stationnaire. Ce quotient est donc la probabilité conditionnelle de quitter S en un pas, en partant de S : Φ mesure la capacité de la chaîne à éviter de s'enfermer dans une petite région de l'espace des états.

Le théorème 21.24 donne une $O(\log n)$-approximation pour le calcul de la conductance. Commençons par remarquer qu'il suffit d'approcher la variante symétrisée suivante de Φ :

$$\Phi' = \min_{S \subset X,\ 0<\pi(S)\leqslant 1} \frac{w(S,\overline{S})}{\pi(S)\pi(\overline{S})}, \tag{21.3}$$

car Φ et Φ' sont à un facteur 2 l'un de l'autre (remarquez que si $0 < \pi(S) \leqslant 1/2$, alors $1/2 \leqslant \pi(\overline{S}) < 1$).

Démontrons maintenant que le calcul de Φ' est un cas particulier du problème de la coupe la moins dense. Reprenons le graphe $G = (X, E)$ muni de poids sur les arêtes, défini précédemment. Associons à toute paire de sommet $x, y \in X$, un flot distinct de demande $\pi(x)\pi(y)$. Il est facile de voir que, pour cette instance, la densité d'une coupe $(S,\overline{S})$ correspond au quotient de l'équation (21.3). Par conséquent, la densité de la coupe la moins dense vaut Φ'.

21.6.3 Coupe équilibrée

L'objet de cette section a des applications dans divers problèmes de partitionnement, tels que le partitionnement de circuits VLSI. De plus, il est utile pour la conception de stratégies de type diviser-pour-régner (pour l'étape de division) ; par exemple, dans l'algorithme pour le problème 21.29 ci-après.

Problème 21.27 (Coupe b-équilibrée minimum)[10] Étant donné un graphe non orienté $G = (V, E)$ muni de coûts positifs sur les arêtes, et un rationnel b, $0 < b \leqslant 1/2$, trouver une coupe de coût minimum $(S,\overline{S})$, telle que $b \cdot n \leqslant |S| < (1-b) \cdot n$.

Une coupe 1/2-équilibrée est appelée une *bisection*, et le problème associé est appelé le problème de la *bisection minimum*.[11] Nous allons utiliser le théorème 21.24 pour obtenir une *pseudo-approximation* pour le problème 21.27 – nous allons construire une coupe 1/3-équilibrée dont le coût est à un facteur $O(\log n)$ du coût de la bisection minimum (se référer aux notes, section 21.8, pour une véritable approximation pour ce problème).

[10] *Minimum b-balanced cut*, en anglais.
[11] *Minimum bisection problem*, en anglais.

Pour $V' \subset V$, notons $G_{V'}$ le sous-graphe de G induit par V'. Voici l'algorithme : commencer avec $U \leftarrow \varnothing$ et $V' \leftarrow V$; tant que $|U| < n/3$, trouver un ensemble W d'expansion à $O(\log n)$ du minimum dans $G_{V'}$, et faire $U \leftarrow U \cup W$ et $V' \leftarrow V' - W$; enfin, poser $S \leftarrow U$ et renvoyer la coupe $(S, V - S)$.

Fait 21.28 *La coupe calculée par l'algorithme est* 1/3*-équilibrée, et son coût est inférieur à* $O(\log n)$ *fois le coût de la bisection minimum de* G*.*

Preuve : À la fin de l'avant-dernière itération, $|U| < n/3$. Donc, au début de la dernière itération, $|V'| \geqslant 2n/3$. Moins de la moitié de ces sommets est ajoutée à U à la dernière itération. Donc, $|V - S| \geqslant n/3$ et $n/3 \leqslant |S| < n/3$. Par conséquent, $(S, V - S)$ est bien une coupe 1/3-équilibrée.

Soit $(T, \overline{T})$ une bisection minimum de G. Puisqu'au début de chaque itération, $|V'| \geqslant 2n/3$, les ensembles $T \cap V'$ et $\overline{T} \cap V'$ contiennent chacun plus de $n/6$ sommets. Ainsi, l'expansion minimum d'un ensemble de sommets de $G_{V'}$ est, à chaque itération, inférieure à $\frac{c(T)}{(n/6)}$. Puisque l'algorithme construit un ensemble d'expansion à un facteur $O(\log n)$ de l'optimale à chaque itération, l'ensemble U calculé vérifie :

$$\frac{c(U)}{|U|} \leqslant O(\log n) \cdot \frac{c(T)}{n/6}.$$

Puisque l'ensemble final S contient moins de $2n/3$ sommets, en sommant les différentes contributions, nous obtenons :

$$c(S) \leqslant O(\log n) \cdot \frac{c(T)(2n/3)}{n/6},$$

c'est-à-dire $c(S) = O(\log n) \cdot c(T)$. □

21.6.4 Arrangement linéaire de coupes minimum

Problème 21.29 (Arrangement linéaire de coupes minimum)[12] Étant donné un graphe non orienté $G = (V, E)$ muni de coûts positifs sur les arêtes, et une numérotation des sommets de 1 à n, notons S_i l'ensemble des sommets numérotés de 1 à i, pour $1 \leqslant i \leqslant n - 1$. Le problème est de trouver une numérotation des sommets, qui minimise le maximum des capacités (c'est-à-dire des coûts) des $n-1$ coupes $(S_i, V - S_i)$, c'est-à-dire qui minimise $\max\{c(S_i) : 1 \leqslant i \leqslant n - 1\}$.

Nous allons utiliser la pseudo-approximation ci-dessus pour le problème de coupe 1/3-équilibrée, et obtenir une véritable $O(\log^2 n)$-approximation

[12] *Minimum cut linear arrangement*, en anglais.

pour notre problème. Une observation essentielle est que $S_{n/2}$ est une bisection, quelle que soit la numérotation, et donc la capacité β d'une bisection minimum de G est un minorant pour l'arrangement optimal. Nous obtiendrons alors un véritable algorithme d'approximation car les performances de l'algorithme pour la coupe 1/3-équilibrée sont mesurées relativement à β.

L'algorithme est récursif : trouver une coupe 1/3-équilibrée $(S, \overline{S})$ dans G_V, et construire récursivement une numérotation de S dans G_S en utilisant les nombres de 1 à $|S|$, et une numérotation de $\overline{S}$ in $G_{\overline{S}}$ en utilisant les nombres de $|S|+1$ à n. Bien sûr, la récursion se termine sur les singletons, par l'attribution du numéro au sommet.

Fait 21.30 *L'algorithme ci-dessus est une $O(\log^2 n)$-approximation pour le problème de l'arrangement linéaire de coupes minimum.*

Preuve : L'arbre binaire T suivant (qui n'est pas nécessairement complet) encode les appels récursifs de l'algorithme. Chaque nœud de l'arbre correspond à un appel récursif. Supposons que l'appel récursif α se termine par deux nouveaux appels récursifs α_1 et α_2, où le premier attribue les numéros les plus petits et le second les plus grands. Alors, α_1 sera le fils gauche de α dans T, et α_2 le fils droit. Si l'appel récursif α a lieu sur un singleton, alors α est une feuille de l'arbre.

Nous associons à chaque nœud interne, l'ensemble des arêtes de la coupe calculée durant cet appel, et à chaque feuille, le sommet du singleton passé en argument. Ainsi, un parcours de gauche à droite des feuilles donne la numérotation des sommets définie par l'algorithme. De plus, les ensembles associés aux nœuds internes définissent une partition de toutes les arêtes de G. D'après le fait 21.28, le coût total des arêtes associées à chaque nœud interne, est $O(\beta \log n)$. Puisque chaque appel récursif construit une coupe 1/3-équilibrée, la profondeur de la récursion, c'est-à-dire la hauteur de T, est bornée par $O(\log n)$.

L'observation suivante sera cruciale. Considérons une arête uv de G. Soit α l'ancêtre commun le plus bas des feuilles associées à u et v dans T. Alors uv appartient à l'ensemble des arêtes associé au nœud α.

Considérons les coupes $(S_i, \overline{S_i})$, $1 \leqslant i \leqslant n-1$, conformément à la numérotation calculée par l'algorithme. Chaque arête de $(S_i, \overline{S_i})$ relie deux sommets numérotés j et k, avec $j \leqslant i$ et $k \geqslant i+1$. Ainsi, une telle arête est associée à un nœud qui est un ancêtre commun des feuilles numérotées i et $i+1$. Puisque la hauteur de T est $O(\log n)$, il y a $O(\log n)$ tels ancêtres. Comme le coût total des arêtes associées à chaque nœud de T est $O(\beta \log n)$, la capacité de la coupe $(S_i, \overline{S_i})$ est inférieure à $O(\beta \log^2 n)$. La preuve se termine en remarquant que β est bien un minorant pour l'arrangement optimal. □

21.7 Exercices

21.1 Pour les trois métriques de l'exemple 21.15, l'espérance de la contribution d'un des ensembles $S_2, \ldots, S_{l+1}$ est toujours en $\Omega(d(u,v)/l)$. Exhibez une métrique dans laquelle l'espérance de la contribution de chacun des ensembles est $\Theta(d(u,v)/l^2)$.

21.2 Démontrez qu'on peut toujours plonger isométriquement n points d'un espace ℓ_1 dans un espace ℓ_2^2 (de plus grande dimension), où la distance entre deux points est le carré de leur distance Euclidienne ℓ_2.
Indication : Comme ℓ_1 et ℓ_2^2 sont toutes deux additives suivant les dimensions, commencez par démontrer qu'il suffit de considérer n points en dimension 1. Indexez ces points $x_1, \ldots, x_n$ par abscisses croissantes. Puis, plongez-les dans $(\mathbf{R}^{n-1}, \ell_2^2)$ comme suit. Posez $\alpha_i = x_{i+1} - x_i$, et envoyez x_i sur $(\sqrt{\alpha_1}, \ldots, \sqrt{\alpha_{i-1}}, 0, \ldots, 0)$.

21.3 Expliquez pourquoi on ne peut pas convertir la pseudo-approximation donnée au début de la section 21.6.3, en une véritable approximation, c'est-à-dire telle qu'à la fin, on puisse comparer la capacité de la coupe 1/3-équilibrée construite à la capacité optimale d'une coupe 1/3-équilibrée.
Indication : Exhibez des graphes pour lesquels la capacité minimum d'une bisection est arbitrairement grande devant celle d'une coupe 1/3-équilibrée.

21.4 Démontrez qu'on peut modifier l'algorithme mentionné ci-dessus pour calculer une coupe b-équilibrée dont la capacité est $O(\log n)$ fois celle de la meilleure coupe b'-équilibrée, pour $b \leqslant 1/3$ et $b < b'$. À quel moment dans l'analyse utilise-t-on $b \leqslant 1/3$?

21.5 Exhibez une réduction isofacteur du problème de la coupe b-équilibrée minimum (avec $b < 1/2$) à celui de la bisection minimum.

21.6 (Linial, London et Rabinovich [198]) Généralisez le théorème 21.22 en démontrant que pour toute métrique $(V, \boldsymbol{d})$ et pour tout $p \geqslant 1$, il existe un plongement ℓ_p de distorsion $O(\log n)$ de $(V, \boldsymbol{d})$ dans un espace de dimension $O(\log^2 n)$.
Indication : Envoyez le point v sur $\left(\frac{d(v,S_i)}{Q^{1/p}}\right)_{i=1,\ldots,Q}$, où Q est la dimension du plongement. Utilisez l'inégalité $|d(u,S_i) - d(v,S_i)| \leqslant d(u,v)$ et la monotonie de la norme ℓ_p.

21.7 (Feige [86]) Étudions l'algorithme suivant pour :

Problème 21.31 (Bande passante minimale)[13] Étant donné un graphe non orienté $G = (V, E)$, numéroter les sommets avec des entiers distincts de

[13] *Bandwidth minimization*, en anglais.

1 à n, de telle sorte que l'étendue maximale d'une arête soit minimale, où l'*étendue* d'une arête uv est la valeur absolue de la différence des numéros attribués à u et à v.

Algorithme 21.32 (Bande passante minimale)

1. Définir la métrique $(V, \boldsymbol{d})$, où d_{uv} est la longueur du chemin le plus court entre u et v dans G.
2. Construire un plongement ℓ_2 de distorsion $O(\log n)$ pour $(V, \boldsymbol{d})$.
3. Tirer une droite aléatoire ℓ selon une distribution à symétrie sphérique, et projeter les n points sur ℓ.
4. Numéroter les sommets de 1 à n dans leur ordre sur ℓ.
5. Renvoyer cette numérotation.

Remarque 21.33 Le lemme 26.7 donne un algorithme pour tirer ℓ aléatoirement.

1. Démontrez que l'espérance du nombre de paires de sommets à distance $\leqslant 1$ l'un de l'autre sur ℓ est :

$$O\left(\log n \sum_{u,v} \frac{1}{d_{uv}}\right).$$

2. Démontrez que

$$\sum_{u,v} \frac{1}{d_{uv}} = O(n \log n \cdot \mathrm{OPT}).$$

   Indication : Remarquez que dans G, le nombre de sommets à distance $\leqslant k$ d'un sommet v est inférieur à $2k \cdot \mathrm{OPT}$.

3. Démontrez qu'avec forte probabilité, l'étendue de la numérotation inversée est inférieure à $O(\sqrt{n\,\mathrm{OPT}} \log n)$, c'est-à-dire cet algorithme est une $O(\sqrt{n} \log n)$-approximation.
   Indication : Si l'étendue de la numérotation renvoyée est s, alors le nombre de paires de sommets à distance $\leqslant 1$ l'un de l'autre sur ℓ est supérieur à s^2.

21.8 Notes

Le résultat fondateur de Leighton et Rao [190] fut à l'origine du premier théorème de flot maximum et de coupe minimum approchés dans le cas du multiflot uniforme. Les auteurs proposèrent également une $O(\log n)$-approximation pour le cas particulier correspondant de la coupe la moins

dense et une pseudo-approximation pour le problème de la coupe b-équilibrée. Klein, Agarwal, Ravi et Rao [181] menèrent la première étude de la généralisation au multiflot sur demande. Le théorème 21.22 est dû à Linial, London et Rabinovich [198], à partir d'un résultat de Bourgain [34] qui démontra l'existence d'un tel plongement et donna un algorithme en temps exponentiel pour le trouver. L'application de ce théorème au problème de la coupe la moins dense, le théorème 21.24, fut trouvée indépendamment par Aumann et Rabani [18], et Linial, London, et Rabinovich [198].

Feige et Krauthgamer proposèrent dans [90] une $O(\log^2 n)$-approximation pour le problème de la bisection minimum, puis pour le problème de la coupe b-équilibrée minimum (voir exercice 21.5). Sinclair [250] proposa d'utiliser le problème de la coupe la moins dense pour évaluer la conductance. L'application des coupes équilibrées au problème de l'arrangement linéaire de coupes fut proposé par Bhatt et Leighton [28]. Se référer à l'exercice 26.9 pour un programme semi-défini pour trouver un plongement ℓ_2^2 de n points ayant une distorsion optimale.

22 Forêt de Steiner

Nous présentons ici une 2-approximation pour le problème de la forêt de Steiner, fondée sur l'augmentation synchrone des solutions duales dans le schéma primal-dual. Le problème de la forêt de Steiner généralise celui de l'arbre de Steiner métrique, pour lequel on avons présenté une 2-approximation au chapitre 3. Nous y avions différé la présentation du minorant sous-jacent à l'algorithme ; nous allons le préciser ici.

À l'instar du problème de l'arbre de Steiner (théorème 3.2) l'étude de la forêt de Steiner repose sur celle du cas métrique (voir exercice 22.2). Mais, comme l'algorithme primal-dual est le même pour les cas métrique et non métrique, nous n'imposons pas de fonction de coût métrique.

Problème 22.1 (Forêt de Steiner)[1] Étant donné un graphe non orienté $G = (V, E)$, muni d'une fonction de coût sur les arêtes $c : E \to \mathbf{Q}^+$, et une collection de sous-ensembles disjoints $S_1, \ldots S_k$ de V, trouver le sous-graphe de coût minimum dans lequel toute paire de sommets d'un même S_i est connectée.

Reformulons le problème ; cela facilitera également sa généralisation par la suite. Introduisons la *fonction requête de connexité* r qui associe 0 ou 1 aux paires de sommets comme suit :

$$r(u, v) = \begin{cases} 1 \text{ si } u \text{ et } v \text{ appartiennent au même ensemble } S_i \\ 0 \text{ sinon} \end{cases}$$

Il s'agit à présent de trouver le sous-graphe F de coût minimum qui contienne un chemin entre u et v pour chaque paire $\{u, v\}$ telle que $r(u, v) = 1$. En général, la solution de ce problème est une forêt.

22.1 La relaxation linéaire et son dual

Pour obtenir un programme linéaire en nombres entiers pour ce problème, nous définissons la fonction $f : 2^V \to \{0, 1\}$ suivante sur les coupes de G, qui indique si une arête traverse la coupe dans toute solution réalisable.

$$f(S) = \begin{cases} 1 \text{ si } \exists\ u \in S \text{ et } v \in \overline{S} \text{ tels que } r(u, v) = 1 \\ 0 \text{ sinon} \end{cases}$$

[1] *Steiner forest*, en anglais.

Associons à chaque arête $e \in E$ une variable x_e, à valeurs dans $\{0,1\}$; x_e vaut 1 ssi e est sélectionnée dans le sous-graphe. Nous obtenons le programme :

$$\begin{aligned} &\text{minimiser} && \sum_{e \in E} c_e x_e && && (22.1)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \in \delta(S)} x_e \geqslant f(S), && S \subseteq V \\ & && x_e \in \{0,1\}, && e \in E \end{aligned}$$

où on note $\delta(S)$ l'ensemble des arêtes qui traversent la coupe $(S, \overline{S})$.

Nous obtenons la relaxation linéaire suivante de (22.1), où nous avons supprimé les conditions redondantes $x_e \leq 1$.

$$\begin{aligned} &\text{minimiser} && \sum_{e \in E} c_e x_e && && (22.2)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \in \delta(S)} x_e \geqslant f(S), && S \subseteq V \\ & && x_e \geq 0, && e \in E \end{aligned}$$

Le programme dual associé est :

$$\begin{aligned} &\text{maximiser} && \sum_{S \subseteq V} f(S) \cdot y_S && && (22.3)\\ &\text{sous les contraintes} && \sum_{S\,:\,e \in \delta(S)} y_S \leq c_e, && e \in E \\ & && y_S \geq 0, && S \subseteq V \end{aligned}$$

Remarquez que les programmes primal et dual forment une paire de programmes couverture-empaquetage (cf. section 13.1).

22.2 Schéma primal-dual synchronisé

Nous complétons le schéma primal-dual avec une nouvelle idée qui le distingue de son usage habituel en algorithmique exacte. Les algorithmes précédents fonctionnent *à la demande* – à chaque itération, une nouvelle condition des écarts complémentaires non satisfaite, est satisfaite en modifiant les solutions primale et duale convenablement. L'idée est ici d'augmenter les solutions duales de façon *synchrone*. L'algorithme ne cherche plus à satisfaire *une* condition en particulier. Il modifie, au contraire, plusieurs variables et conditions simultanément, et l'une de ces modifications améliore la solution primale.

Un peu de vocabulaire facilitera la description de l'algorithme. Nous dirons que l'arête e est *touchée*[2] par la variable duale y_S si $y_S > 0$ et $e \in \delta(S)$. Nous dirons que l'ensemble S a été *levé*[3] par la solution duale si $y_S > 0$. Lever S ou $\overline{S}$ revient clairement au même, aussi, nous dirons simplement qu'on a levé la coupe $(S, \overline{S})$. Remarquons qu'il n'y a aucun intérêt à lever un ensemble S tel que $f(S) = 0$ puisqu'il ne contribue pas à la valeur objectif duale. Nous supposerons donc que de telles coupes ne sont jamais levées. Nous dirons qu'une arête e est *saturée*[4] si son coût est égal à la somme des variables duales qui la touchent. Le programme dual cherche à maximiser la somme des variables duales y_S sous la contrainte qu'aucune arête ne doit être touchée par la solution duale à un niveau supérieur à son coût, c'est-à-dire aucune arête ne doit *déborder*.

Établissons maintenant les conditions des écarts complémentaires relâchées recherchées. Toute arête sélectionnée par l'algorithme l'est intégralement. Nous appellerons *degré de l'ensemble* S, le nombre d'arêtes sélectionnées traversant la coupe $(S, \overline{S})$.

Conditions primales : Pour tout $e \in E$, $x_e \neq 0 \Rightarrow \sum_{S\,:\,e\in\delta(S)} y_S = c_e$. C'est-à-dire, *toute arête sélectionnée est saturée.*

Conditions duales relâchées : La relaxation suivante des conditions duales devrait conduire à une 2-approximation : pour tout $S \subseteq V, y_S \neq 0 \Rightarrow \sum_{e\,:\,e\in\delta(S)} x_e \leqslant 2 \cdot f(S)$, c'est-à-dire le degré de toute coupe levée est inférieur à 2. Cependant, nous ne savons pas comment garantir cette condition. De façon surprenante, nous obtenons une 2-approximation malgré tout, en relâchant encore cette condition ! Les ensembles levés sont autorisés à prendre un degré arbitraire, mais *le degré moyen des ensembles levés doit être inférieur à* 2. La définition exacte de « degré moyen » sera donnée plus tard.

L'algorithme commence avec deux solutions primale et duale nulles. Dans l'esprit du schéma primal-dual, la solution primale courante indique quelle coupe lever, alors que la solution duale indique quelle arête sélectionner. Chaque itération de l'algorithme améliore la réalisabilité de la solution primale et la valeur de la solution duale, jusqu'à ce que la primale soit réalisable.

Décrivons à présent le déroulement de chaque itération. À tout instant, l'ensemble des arêtes sélectionnées forme une forêt. Nous dirons qu'un ensemble S est *insatisfait* si $f(S) = 1$ et si aucune arête traversant la coupe $(S, \overline{S})$ n'est sélectionnée. Un ensemble S est dit *actif* si c'est un ensemble insatisfait minimal (pour l'inclusion) dans l'itération courante. Si la solution primale courante n'est pas réalisable, il existe nécessairement un ensemble insatisfait, et donc un ensemble actif.

[2] *e feels dual* y_S, en anglais.
[3] *S is raised in a dual solution*, en anglais.
[4] *Tight*, en anglais.

Lemme 22.2 *Un ensemble S est actif ssi S est une composante connexe de la forêt sélectionnée courante, telle que $f(S) = 1$.*

Preuve : Soit S un ensemble actif. S ne peut contenir un sous-ensemble strict d'une composante connexe de la forêt sélectionnée, sinon une arête sélectionnée appartiendrait à la coupe $(S, \overline{S})$. S est donc une union de composantes connexes de la forêt. Comme $f(S) = 1$, il existe deux sommets $u \in S$ et $v \in \overline{S}$ tels que $r(u, v) = 1$. Soit S' la composante connexe de la forêt contenant u. Clairement, S' est également insatisfait et par minimalité de S, $S = S'$. □

On trouve facilement tous les ensembles actifs de l'itération courante avec la caractérisation du lemme 22.2. Les variables duales de ces ensembles sont toutes augmentées de façon synchrone, jusqu'à ce qu'une arête soit saturée. Toutes les arêtes nouvellement saturées sont sélectionnées, ce qui termine l'itération courante.

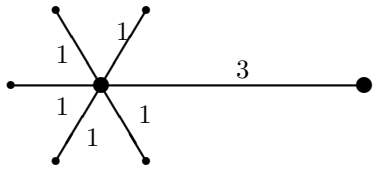
Quand une solution réalisable primale est atteinte, notons-la F, l'étape de sélection des arêtes se termine. Mais F peut contenir des arêtes superflues qui doivent être éliminées pour garantir le facteur d'approximation ; cette étape est illustrée dans l'exemple 22.4. Formellement, une arête $e \in F$ est dite *redondante* si $F \setminus \{e\}$ est une solution réalisable. Toutes les arêtes redondantes peuvent être éliminées simultanément de F. De manière équivalente, seules les arêtes non redondantes sont conservées.

L'algorithme est décrit ci-dessous. Nous laissons en exercice la conception d'une implémentation efficace.

Algorithme 22.3 (Forêt de Steiner)

1. **(Initialisation)** $F \leftarrow \varnothing$; pour tout $S \subseteq V$, $y_S \leftarrow 0$.
2. **(Augmentation des arêtes)**
   Tant qu'il existe un ensemble insatisfait faire :
   Augmenter simultanément les y_S pour les S actifs, jusqu'à ce qu'une des arêtes e soit saturée ;
   $F \leftarrow F \cup \{e\}$.
3. **(Élimination)** Renvoyer $F' = \{e \in F : F - \{e\}$ n'est pas réalisable$\}$

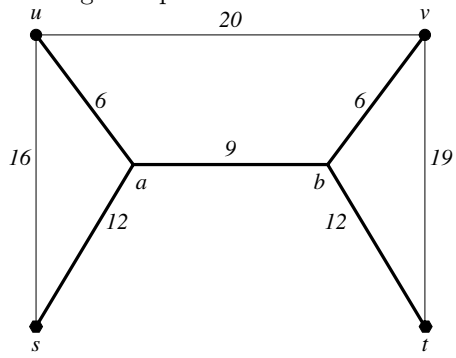
Exemple 22.4 Considérons une étoile où toutes les arêtes coûtent 1, sauf une qui coûte 3.



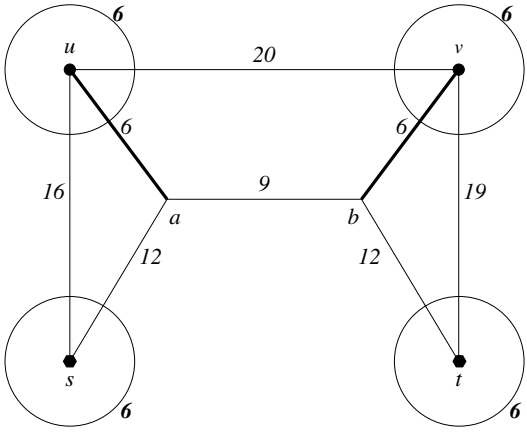
L'unique contrainte est de connecter les extrémités de l'arête de coût 3. L'algorithme va sélectionner dans F toutes les arêtes de coût 1 avant de sélectionner l'arête de coût 3. Clairement, à cet instant, F n'est pas à un facteur 2 de l'optimum. Cependant, ceci est corrigé par l'étape d'élimination qui supprime toutes les arêtes de coût 1. □

Exécutons maintenant l'algorithme sur un exemple non trivial qui illustre son fonctionnement en détail.

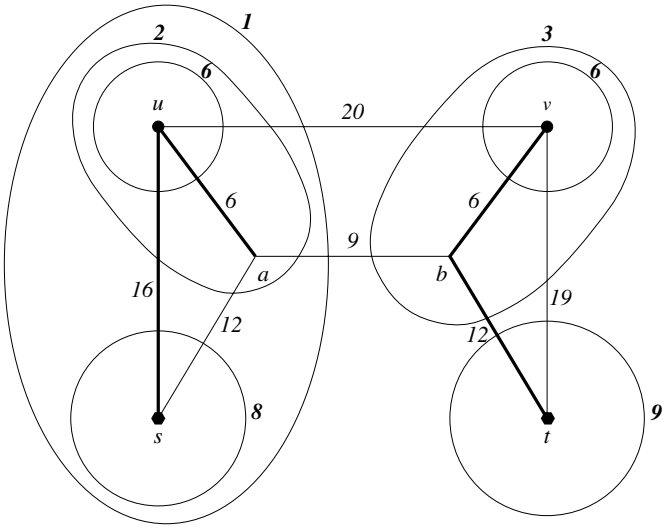
Exemple 22.5 Étudions le graphe suivant, où les coûts sont indiqués sur les arêtes. Les seules contraintes non nulles de connexité sont $r(u, v) = 1$ et $r(s, t) = 1$. Les arêtes en gras représentent une solution de coût optimal 45.



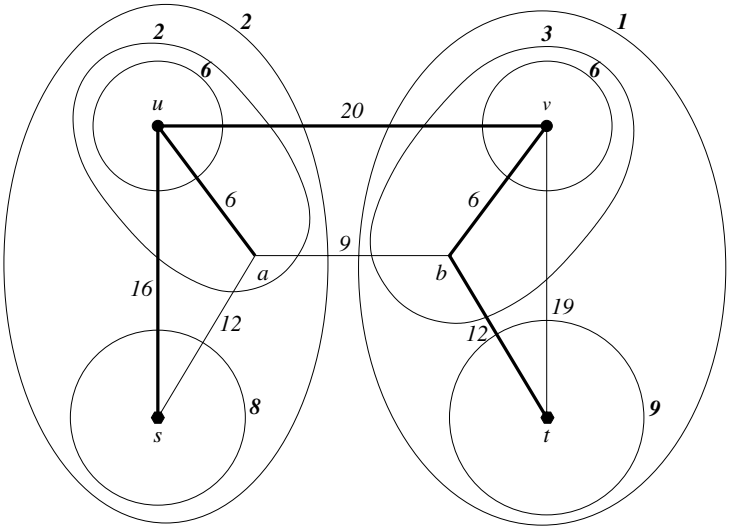
Lors de la première itération, les ensembles actifs sont les quatre singletons suivants : $\{s\}$, $\{t\}$, $\{u\}$ et $\{v\}$. Leurs variables duales sont portées à 6, et les arêtes ua et vb deviennent saturées. L'une d'entre elles, disons ua, est sélectionnée et l'itération se termine. Durant la deuxième itération, $\{u, a\}$ devient actif à la place de $\{u\}$. Il est inutile d'augmenter les variables duales à cette itération, car il y a déjà une arête saturée vb. Cette arête vb est sélectionnée et l'itération se termine. Les solutions primales et duales à cet instant sont données ci-dessous (les arêtes sélectionnées sont en gras) :



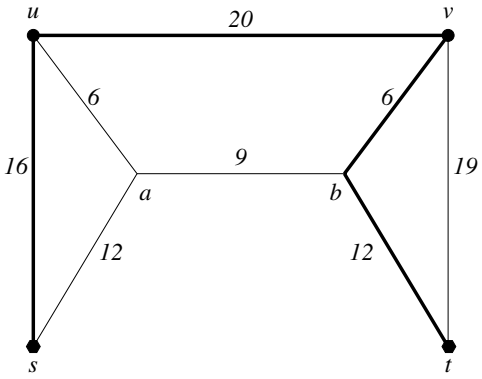
Lors de la troisième itération, $\{v, b\}$ devient actif à la place de $\{v\}$. Après que les ensembles actifs aient été augmentés de 2 chacun, l'arête us, nouvellement saturée, est sélectionnée. À la quatrième itération, les ensembles actifs sont $\{u, s, a\}, \{v, b\}$ et $\{t\}$. Une fois qu'ils ont tous été augmentés de 1, l'arête bt, nouvellement saturée, est sélectionnée. Voici la situation courante :



À la cinquième itération, les ensembles actifs sont $\{a, s, u\}$ et $\{b, v, t\}$. Ils sont tous augmentés de 1, et l'arête uv est saturée puis sélectionnée. Nous avons alors obtenu une solution primale réalisable :



Durant l'étape d'élimination, l'arête *ua* est supprimée et on obtient la solution suivante coûtant 54 :



□

22.3 Analyse

Le lemme 22.6 garantit que la solution primale reste réalisable après l'élimination *simultanée* de toutes les arêtes redondantes, c'est-à-dire il est impossible que deux arêtes soient toutes deux redondantes, et que l'élimination de l'une rende l'autre non redondante.

Lemme 22.6 *À la fin de l'algorithme, F' et $\boldsymbol{y}$ sont des solutions réalisables primale et duale, respectivement.*

Preuve : À la fin de l'étape 2, F satisfait toutes les contraintes de connexité. Lors de chaque itération, seules les variables duales des composantes connexes sont augmentées. Aucune arête interne à une composante ne peut donc être saturée, et F est acyclique, c'est-à-dire F est une forêt. Par conséquent, si $r(u,v) = 1$, il existe un *unique* chemin entre u et v dans F. Ainsi, aucune arête de ce chemin n'est redondante et aucune ne sera donc éliminée lors de l'étape 3. F' est donc bien une solution primale réalisable.

Dès qu'une arête est saturée, l'itération courante prend fin. Comme les ensembles actifs sont reconstruits, aucune arête ne déborde jamais. Nous en concluons que $\boldsymbol{y}$ est une solution duale réalisable. □

Notons $\deg_{F'}(S)$ le nombre d'arêtes de F' traversant la coupe $(S, \overline{S})$. La propriété suivante, vérifiée par les degrés des composantes satisfaites, est la clé pour obtenir le facteur d'approximation.

Lemme 22.7 *À chaque itération de l'algorithme, pour toute composante connexe C du sous-graphe induit par les arêtes courantes sélectionnées : si $f(C) = 0$ alors $\deg_{F'}(C) \neq 1$.*

Preuve : Supposons que $\deg_{F'}(C) = 1$ et notons e l'unique arête de F' traversant la coupe $(C, \overline{C})$. Puisque e n'est pas redondante (aucune arête de F' n'est redondante), il existe une paire de sommets $\{u, v\}$, telle que $r(u,v) = 1$ et telle que e appartienne à l'unique chemin entre u et v dans F'. Comme ce chemin ne traverse la coupe $(C, \overline{C})$ qu'une seule fois, l'un des sommets appartient à C et l'autre à $\overline{C}$. Or puisque $r(u,v) = 1$, nous avons $f(C) = 1$, CQFD. □

Lemme 22.8 $$\sum_{e \in F'} c_e \leq 2 \sum_{S \subseteq V} y_S$$

Preuve : Comme toutes les arêtes sélectionnées sont saturées,

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \left(\sum_{S \,:\, e \in \delta(S)} y_S \right).$$

Permuter les sommes donne :

$$\sum_{e \in F'} c_e = \sum_{S \subseteq V} \left(\sum_{e \in \delta(S) \cap F'} y_S \right) = \sum_{S \subseteq V} \deg_{F'}(S) \cdot y_S.$$

Il suffit donc de montrer que :

$$\sum_{S \subseteq V} \deg_{F'}(S) \cdot y_S \leq 2 \sum_{S \subseteq V} y_S. \tag{22.4}$$

Nous allons démontrer le fait plus fort suivant : à chaque itération, la variation du membre de gauche de l'inégalité (22.4) est majorée par celle du membre de droite. Notons Δ la variation des variables associées aux ensembles actifs durant une itération donnée. Il suffit de montrer que

$$\Delta \times \left(\sum_{S \text{ actif}} \deg_{F'}(S) \right) \leq 2\Delta \times (\text{nombre d'ensembles actifs}).$$

Remarquons que le degré (calculé sur F') de tout ensemble actif S n'est dû qu'aux arêtes sélectionnées *pendant ou après* l'itération courante. Réécrivons l'inégalité ainsi :

$$\frac{\sum_{S \text{ actif}} \deg_{F'}(S)}{\text{nombre d'ensembles actifs}} \leq 2. \tag{22.5}$$

Il suffit donc de montrer que durant cette itération, le degré moyen des ensembles actifs (calculé sur F') est inférieur à 2. Ceci découle mécaniquement du fait que dans un arbre, et plus généralement dans une forêt, le degré moyen est inférieur à 2.

Notons H le graphe ayant pour sommets V et arêtes F'. Considérons l'ensemble des composantes connexes dans F au début de l'itération courante. Construisons le graphe H' à partir de H, en fusionnant dans H chacune de ces composantes connexes en un sommet unique. Remarquons que de H à H', toutes les arêtes sélectionnées dans F avant l'itération courante ont été contractées. Par construction, le degré d'un sommet de H' est égal au degré de la composante associée dans H. Nous dirons qu'un nœud de H' est *actif* si sa composante associée est active ; les autres nœud seront dit *inactifs*. Le degré de tout nœud actif de H' est non nul (puisqu'il doit avoir une arête incidente pour satisfaire sa contrainte de connexité), et H' est une forêt. Éliminons tous les nœuds isolés de H'. Le graphe restant est une forêt de degré moyen inférieur à 2. D'après le lemme 22.7, le degré de tout nœud inactif de ce graphe est au moins 2, c'est-à-dire la forêt n'a pas de feuilles inactives. Par conséquent, le degré moyen des nœuds actifs est bien inférieur à 2. □

Remarquez que la preuve utilise un argument de type coûts amortis[5] : il existe de nombreuses feuilles actives dans la forêt qui contrebalancent tout nœud actif de degré supérieur à 2. Les conditions des écarts complémentaires duales relâchées sont maintenant correctement définies et vérifiées : à chaque itération, les variables duales augmentées ont un degré moyen inférieur à 2. Les lemmes 22.6 et 22.8 permettent alors de conclure :

Théorème 22.9 *L'algorithme 22.3 est une* 2*-approximation pour le problème de la forêt de Steiner.*

[5] *Charging argument*, en anglais.

L'instance critique proposée pour l'arbre de Steiner métrique (ex. 3.4) est également critique pour cet algorithme. L'algorithme 22.3 démontre que le saut intégral de la relaxation linéaire (22.2) pour le problème de la forêt de Steiner est inférieur à 2. L'exemple 22.10 minore ce saut par 2 (asymptotiquement), même pour la recherche d'un arbre couvrant minimum.

Exécutons l'algorithme 22.3 sur une instance du problème de l'arbre de Steiner métrique. Si les arêtes vérifient strictement l'inégalité triangulaire, c'est-à-dire si pour tout triplet de sommets (u, v, w), $c(u, v) < c(u, w) + c(v, w)$, alors on peut voir facilement que l'algorithme trouve un arbre couvrant de poids minimum[6] des sommets requis : on retrouve essentiellement l'algorithme pour l'arbre de Steiner décrit au chapitre 3. Lorsque l'inégalité triangulaire est large, le coût de la solution calculée est identique à celui d'un MST. De plus, si l'algorithme sélectionne toujours de préférence les arêtes reliant des sommets requis parmi les arêtes saturées, il construira bien un MST. Nous avons donc bien retrouvé le minorant sur lequel cet algorithme est bâti.

Le problème du MST est également un cas particulier résolu par cet algorithme : celui où toutes les paires de sommets doivent être connectées. En effet, sur ces instances, l'algorithme 22.3 exécute essentiellement l'algorithme de Kruskal : il sélectionne à chaque itération l'arête la moins chère reliant deux composantes connexes, et calcule donc un MST. Remarquez cependant que, comme le démontre l'exemple 22.10, la solution duale obtenue peut ne valoir que la moitié de l'optimal.

Exemple 22.10 Considérons un cycle à n sommets, avec des coûts unitaires sur les arêtes. Le coût d'un MST est $n - 1$. Mais la solution duale obtenue par l'algorithme 22.3 vaut $n/2 : 1/2$ autour de chaque sommet. Cette solution est optimale puisqu'elle est réalisée par la solution primale fractionnaire qui sélectionne chaque arête à moitié. Cet exemple minore donc par 2 (asymptotiquement) le saut intégral du programme linéaire (22.2), même s'il est restreint au problème de l'arbre couvrant de poids minimum. □

22.4 Exercices

22.1 Démontrez à l'aide du théorème du flot maximum et de la coupe minimum qu'un sous-graphe de G contient tous les chemins requis ssi il satisfait toutes les contraintes de coupes du programme entier (22.1). En déduire que le programme en nombres entiers (22.1) résout bien le problème de la forêt de Steiner.

22.2 Exhibez une réduction isofacteur du problème de la forêt de Steiner à celui de la forêt de Steiner métrique. Démontrez qu'on peut, sans perte de

[6] *Minimum spanning tree (MST)*, en anglais.

généralité, imposer que les coûts des arêtes vérifient l'inégalité triangulaire pour le problème de forêt de Steiner.
Indication : Le raisonnement est le même que pour le problème de l'arbre de Steiner.

22.3 Quels seraient la réalisabilité de la solution obtenue et le facteur d'approximation si

1. l'étape d'élimination de l'algorithme 22.3 était remplacée par l'étape d'élimination en arrière de l'algorithme 18.4.
2. l'étape d'élimination en arrière de l'algorithme 18.4 était remplacée par l'étape d'élimination de l'algorithme 22.3.

22.4 Exhibez une instance sur laquelle l'algorithme 22.3 augmente une coupe de degré supérieur à 3 (le degré étant calculé sur le sous-graphe correspondant à la solution primale obtenue après la phase d'élimination).

22.5 Lancez l'algorithme 22.3 sur une instance du problème de l'arbre couvrant de poids minimum. Prenez un sommet arbitraire pour racine, et éliminez toutes les variables duales non nulles qui contiennent ce sommet. Démontrez que le coût de l'arbre obtenu est égal au double de la somme des variables duales restantes.
Indication : Démontrez que pour toute itération qui commence avec k composantes connexes et augmente les variables duales de Δ, le membre de gauche de l'inégalité (22.4) augmente exactement de $2(k-1)\Delta$.

22.6 Donnons-nous une échelle de temps continue durant l'étape 2 de l'algorithme 22.3, c'est-à-dire supposons que les variables duales augmentent d'une unité par unité de temps. Considérons $\langle G = (V, E), c, S_1, \ldots, S_k \rangle$, une instance du problème de la forêt de Steiner, ainsi qu'une variante de cette instance où un des sommets de $V - (S_1 \cup \ldots \cup S_k)$ est ajouté à l'un des S_i. Lançons l'algorithme 22.3 sur ces deux instances et notons respectivement R_1 et R_2 ces deux exécutions.

1. Démontrez que si $k = 1$, c'est-à-dire si l'instance posée est une instance de l'arbre de Steiner, alors : si deux sommets $u, v \in S_1$ sont connectés par un chemin saturé dans l'exécution R_1 à l'instant t, alors u et v sont également connectés par un chemin saturé dans R_2 au même instant.
2. Exhibez un contre-exemple pour $k > 1$.

22.7 (Goemans et Williamson [112]) L'algorithme 22.3 fonctionne en fait sur une classe plus générale de problèmes dont la forêt de Steiner est un cas particulier. Nous dirons ici qu'une fonction $f : 2^V \to \{0, 1\}$ est *propre*[7] si elle vérifie les propriétés suivantes :

1. $f(V) = 0$;

[7] *Proper function*, en anglais.

2. $f(S) = f(\overline{S})$;
3. si A et B sont deux sous-ensembles disjoints de V et $f(A \cup B) = 1$, alors $f(A) = 1$ ou $f(B) = 1$.

Remarquez que la fonction f introduite pour le problème de la forêt de Steiner est propre. Considérons la famille des programmes en nombres entiers (22.1) où f est une fonction propre quelconque. Démontrez que l'algorithme 22.3 est une 2-approximation pour cette classe de programmes entiers.

22.8 (Goemans et Williamson [112]) Étudions le problème suivant.

Problème 22.11 (Connexion point-à-point)[8] Étant donné un graphe $G = (V, E)$, muni d'une fonction de coût sur les arêtes $c : E \to \mathbf{Q}^+$ (pas nécessairement métrique) et deux ensembles disjoints de sommets S et T de même cardinaux, trouver un sous-graphe de coût minimum tel qu'il existe des chemins reliant tout sommet de S à un unique sommet de T.

1. Donnez une 2-approximation pour ce problème.
   Indication : Montrez que ce problème s'exprime comme un programme linéaire en nombres entiers de type (22.1), où f est une fonction propre.
2. Considérez la relaxation de ce problème qui demande que chaque sommet de S soit relié à un sommet de T (pas nécessairement unique). Proposez également une 2-approximation pour ce problème.
   Indication : Réduisez-le au problème de l'arbre de Steiner.

22.9 (Goemans et Williamson [112]) Étudions la variante suivante du problème de l'arbre de Steiner métrique.

Problème 22.12 (Arbre de Steiner à péage)[9] Étant donné un graphe complet non orienté $G = (V, E)$, un sommet distingué $r \in V$, une fonction métrique coût : $E \to \mathbf{Q}^+$ sur les arêtes, et une fonction de péage[10] $\pi : V \to \mathbf{Q}^+$ sur les sommets. Il s'agit de trouver un arbre contenant r, qui minimise la somme des coûts des arêtes sélectionnées et des péages des sommets non sélectionnés.

1. Considérons le programme entier 22.6 suivant qui associe une variable x_e à chaque arête e et une variable Z_T à chaque sous-ensemble de sommets T qui ne contient pas r. Z_T vaudra 1 sur l'ensemble T des sommets non sélectionnés dans l'arbre optimal.

[8] *Point-to-point connection*, en anglais.
[9] *Prize-collecting Steiner tree*, en anglais.
[10] *Penalty function*, en anglais.

$$\text{minimiser} \quad \sum_{e\in E} c_e x_e + \sum_{T\subseteq V\,:\, r\notin T} \left(Z_T \sum_{v\in T} \pi_v \right) \tag{22.6}$$

$$\text{sous les contraintes} \quad \sum_{e\in\delta(S)} x_e + \sum_{T\supseteq S} Z_T \geqslant 1, \qquad S \subseteq V ; r \notin S$$

$$x_e \in \{0,1\}, \qquad e \in E$$

$$Z_T \in \{0,1\}, \qquad T \subseteq V ; r \notin T$$

Déterminez sa relaxation linéaire et son dual. Le programme dual associe une variable y_S à chaque sous-ensemble de sommets S qui ne contient pas r.

2. L'algorithme primal-dual suivant pour ce problème est construit sur le même principe que l'algorithme 22.3. Au départ, les *ensembles actifs* sont les singletons $\{v\}$ auxquels sont associées des *charges* π_v. Une liste ordonnée vide F est créée. Les variables duales de chaque ensemble actif sont augmentées de façon synchrone. Au fur et à mesure qu'une variable duale augmente, sa charge diminue d'autant. Dès que la charge d'un ensemble S tombe à zéro, il est déclaré *mort* et tous ses sommets non étiquetés sont étiquetés S. Dès qu'une arête e est saturée, elle est ajoutée à la liste F. Suivant les cas, les actions suivantes sont déclenchées :
   - Si e relie un ensemble actif S à r : l'ensemble S est désactivé et déclaré *connecté à* r ; tous ses sommets non étiquetés, sont étiquetés r.
   - Si e relie un ensemble actif S à un ensemble qui est connecté à r : on applique la même procédure que dans le cas précédent.
   - Si e relie deux ensembles S et S' qui sont, soit tous deux actifs, soit l'un actif et l'autre mort : les ensembles actifs parmi S et S' sont désactivés, et $S \cup S'$ est déclaré actif et reçoit la somme des charges restantes de S et S'.

   Quand il n'existe plus d'ensembles actifs, l'algorithme effectue une étape d'*élimination en arrière dynamique*[11] sur F. Il s'agit d'un raffinement de l'élimination en arrière où les contraintes évoluent dynamiquement. Tous les sommets étiquetés r sont déclarés *requis*. Nous dirons que F est *réalisable* s'il existe un chemin de tout sommet requis à r utilisant les arêtes de F. Pour toute arête $e = uv \in F$, nous appelons *ensemble mort maximal de e contenant v*, l'ensemble S maximal tel que $v \in S$, $u \notin S$ et S ait été déclaré mort par l'algorithme. Si aucun ensemble ne satisfait ces conditions, l'ensemble mort maximal est $\varnothing$.

   Les arêtes $e \in F$ sont prises par date d'insertion décroissante dans F. Pour chaque arête e, si $F \setminus e$ est réalisable, alors e est éliminée de F. Sinon, posons $e = uv$ et notons S l'ensemble mort maximal de e contenant v. Si $S \neq \varnothing$, alors tous les sommets étiquetés S sont déclarés requis. Et on itère sur l'ensemble mort maximal de e contenant u.

[11] *Dynamic reverse delete*, en anglais.

3. Démontrez qu'au départ de l'étape d'élimination en arrière, F est réalisable. Puis que F reste réalisable, même lorsque le nombre de sommets requis augmente.
4. Démontrez qu'à la fin de l'algorithme, F est un arbre contenant r et vérifiant :
   a) F contient tous les sommets étiquetés r ; et
   b) si F contient un sommet étiqueté S , alors F contient aussi tous les sommets étiquetés T , pour $T \supseteq S$.
5. La solution primale renvoyée est construite ainsi : les variables x_e avec $e \in F$ sont mises à 1, et les autres à 0 ; les variables Z_T correspondant aux ensembles maximaux morts T qui ne contiennent aucun sommet de F sont mises à 1, et les autres variables Z à 0.

   Démontrez que cet algorithme est une 2-approximation pour le problème de la forêt de Steiner à péage, en montrant que les solutions primale et duale obtenues vérifient :

$$\sum_{e \in E} c_e x_e + 2 \cdot \sum_{T \subseteq V \,:\, r \notin T} \left(Z_T \sum_{v \in T} \pi_v \right) \leqslant 2 \cdot \sum_{S \subseteq V \,:\, r \notin T} y_S .$$

   Indication : Si $Z_T = 1$, la charge totale des sommets de T est égale à la contribution de T à la valeur du dual. Le coût de l'arbre est alors inférieur au double de la valeur restante du dual. Procédez en démontrant, comme au lemme 22.7, qu'à chaque instant il existe au plus un ensemble inactif de degré 1 (celui qui contient r).

22.10 Étudions la généralisation suivante du problème de la forêt de Steiner avec des contraintes de connexité plus fortes : les contraintes de connexité sont maintenant données par une fonction r qui associe à chaque paire de sommets $\{u, v\}$ le nombre de chemins $r(u, v) \in \{0, \ldots, k\}$ requis entre eux, où k fait partie l'entrée. Chaque arête peut être utilisée plusieurs fois ; chaque copie d'une arête e coûte $c(e)$. Utilisez l'algorithme 22.3 comme une procédure auxiliaire pour construire une $2 \cdot (\lfloor \log_2 k \rfloor + 1)$-approximation pour le problème de la recherche d'un sous-graphe de coût minimum satisfaisant toutes les contraintes de connectivité.

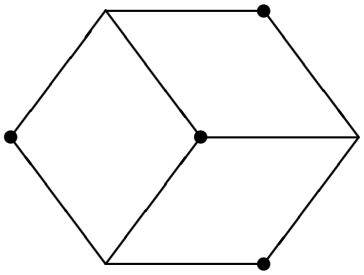
22.11 Nous présentons ci-dessous une *relaxation par coupes orientées*[12] pour le problème de l'arbre de Steiner. On conjecture que cette relaxation a un saut intégral inférieur à la relaxation non orientée (22.2), bien qu'on n'en connaisse pas de preuve aujourd'hui. Nous construisons un graphe orienté H à partir du graphe G en remplaçant chaque arête uv par deux arcs $(u \to v)$ et $(v \to u)$, ayant chacun le même coût que uv. Choisissons un sommet requis

[12] *Bidirected cut relaxation for the Steiner tree problem*, en anglais.

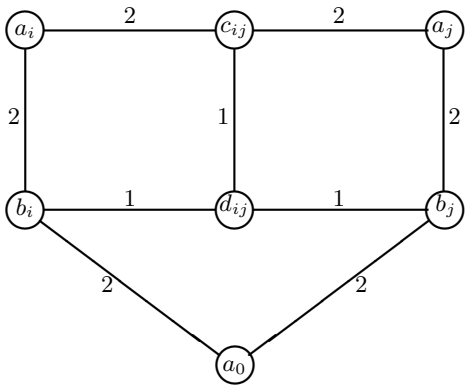
arbitraire r pour racine. Nous dirons que $S \subset V$ est *valide* s'il contient un sommet requis et si $r \in \overline{S}$. Associons une variable indicatrice x_e à chaque arc $e \in H$. Le programme entier est :

$$\begin{aligned} &\text{minimiser} && \sum_{e \in E} c_e x_e && (22.7)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \in \delta(S)} x_e \geqslant 1, && \text{pour tout ensemble valide } S\\ & && x_e \in \{0,1\}, && e \in H \end{aligned}$$

1. Montrez qu'une solution optimale de ce programme entier est un arbre de Steiner optimal.
2. Relaxez le programme entier (22.7) linéairement et donnez son dual.
3. Démontrez que le coût d'une solution optimale de (22.7) et de sa relaxation est indépendant de la racine choisie.
4. Montrez que le saut intégral de la relaxation est inférieur à 2.
5. (Rajagopalan et Vazirani [235]) Montrez que le saut intégral de cette relaxation est 10/9 pour le graphe suivant, où les sommets requis sont représentés par un disque noir.



6. (M. Goemans) La famille de graphes suivante minore le saut intégral de la relaxation (22.7) par 8/7 asymptotiquement. C'est actuellement la pire famille d'instances connue. Le graphe G_n a $n+1$ sommets requis $a_0, a_1, \ldots, a_n$ et n^2 sommets Steiner $b_1, \ldots, b_n$, c_{ij} et d_{ij} pour $1 \leqslant i < j \leqslant n$. La figure ci-dessous décrit les arêtes du graphe ainsi que leurs coûts.



Vérifiez que l'arbre de Steiner optimal coûte $4n$ et qu'une solution optimale de la relaxation (22.7) coûte $7n + 1/2$.

7. Exhibez d'autres graphes pour lesquels cette relaxation admet un saut intégral strictement supérieur à 1 (ce n'est pas facile !).
8. (Edmonds [77]) Étudions le cas particulier où tous les sommets sont requis, c'est-à-dire où l'on recherche un arbre couvrant de poids minimum (MST) de G. Donnez un algorithme primal-dual qui utilise cette relaxation pour trouver un arbre et un dual de même coût, démontrant ainsi que cette relaxation est exacte, c'est-à-dire admet toujours une solution optimale entière, pour le problème de l'arbre couvrant de poids minimum. La relaxation non orientée a en revanche un saut intégral de 2, même pour le problème de l'arbre couvrant de poids minimum.

22.12 (Prömel et Steger [232]) Cet exercice propose un algorithme pour l'arbre de Steiner en utilisant le problème de l'appariement pondéré dans un matroïde[13] et le fait structurel suivant. Nous dirons qu'un arbre de Steiner est *3-contraint* si tout sommet Steiner utilisé dans cet arbre a exactement trois voisins et que ceux-ci sont tous des sommets requis. Le coût optimal d'un arbre de Steiner 3-contraint est inférieur à 5/3 fois le coût d'un arbre de Steiner optimal (Zelikovsky [272]). Démontrez qu'un arbre de Steiner 3-contraint peut être trouvé en temps polynomial à l'aide d'un oracle pour le problème de l'appariement pondéré dans un matroïde. On ne sait pas si ce dernier problème appartient à **P**, ni s'il est **NP**-difficile. Cependant, il existe un algorithme randomisé polynomial, lorsque les poids sont unitaires. Utilisez ce fait et une mise à l'échelle pour obtenir une $(5/3+\varepsilon)$-approximation pour le problème de l'arbre de Steiner, pour tout $\varepsilon > 0$.

Le *problème de l'appariement pondéré dans un matroïde* est défini ainsi. Soit $(S, \mathcal{I})$ un matroïde, où S est l'ensemble support et $\mathcal{I}$ la collection des ensembles indépendants. S est muni de poids positifs sur ses éléments. Les éléments de S sont également partitionnés en paires

[13] *Matroid parity problem*, en anglais.

$\{x_1, x_2\}, \ldots, \{x_{2n-1}, x_{2n}\}$. Le problème consiste à trouver un ensemble de paires de poids maximum, tel que l'ensemble des éléments sélectionnés forme un indépendant.

22.5 Notes

Ce chapitre est construit sur le travail de Goemans et Williamson [112]. La première 2-approximation pour le problème de la forêt de Steiner est due à Agrawal, Klein et Ravi [1]. On pourra également se référer à l'état de l'art de Goemans et Williamson [114].

23 Réseau de Steiner

Connue également sous le nom de *conception de réseaux robustes*,[1] la généralisation suivante du problème de la forêt de Steiner imposant des contraintes de connexité plus fortes, a des applications pour la conception de réseaux. Dans ce chapitre, nous donnons une 2-approximation pour ce problème, fondée sur un raffinement de l'arrondi en programmation linéaire : l'*arrondi répété*. Notons que l'exercice 22.10 traite un cas particulier de ce problème.

Problème 23.1 (Réseau de Steiner)[2] Considérons un graphe non orienté $G = (V, E)$, muni d'une fonction de coût sur les arêtes $c : E \to \mathbf{Q}^+$ (pas nécessairement métrique), des requêtes de connexité r associant un entier positif à chaque paire de sommets, et une fonction $u : E \to \mathbf{N} \cup \{\infty\}$ indiquant le nombre maximum de fois qu'une arête e peut être utilisée (il n'y a pas de borne sur e si $u_e = \infty$). Le problème est de trouver un multigraphe de coût minimum sur les sommets V tel qu'il existe $r(uv)$ chemins arête-disjoints entre toute paire de sommets $u, v \in V$. Chaque copie d'une arête e coûte $c(e)$.

23.1 Relaxation linéaire et solutions demi-entières

Pour obtenir un programme linéaire en nombres entiers, nous définissons la *fonction de contraintes de coupe*[3] $f : 2^V \to \mathbf{N}$ comme pour la forêt de Steiner. Pour tout $S \subseteq V$, $f(S)$ est la plus grande requête de connexité traversant la coupe $(S, \overline{S})$, c'est-à-dire $f(S) = \max\{r(u, v) : u \in S \text{ et } v \in \overline{S}\}$.

[1] *Survivable network design*, en anglais.
[2] *Steiner network*, en anglais.
[3] *Cut requirement function*, en anglais.

$$\begin{aligned}
&\text{minimiser} && \sum_{e\in E} c_e x_e && && (23.1)\\
&\text{sous les contraintes} && \sum_{e\,:\,e\in\delta(S)} x_e \geqslant f(S), && S\subseteq V\\
& && x_e \in \mathbf{N}, && e\in E \text{ et } u_e=\infty\\
& && x_e \in \{0,1,\ldots,u_e\}, && e\in E \text{ et } u_e\neq\infty
\end{aligned}$$

Sa relaxation linéaire est :

$$\begin{aligned}
&\text{minimiser} && \sum_{e\in E} c_e x_e && && (23.2)\\
&\text{sous les contraintes} && \sum_{e\,:\,e\in\delta(S)} x_e \geqslant f(S), && S\subseteq V\\
& && x_e \geqslant 0, && e\in E \text{ et } u_e=\infty\\
& && u_e \geqslant x_e \geqslant 0, && e\in E \text{ et } u_e\neq\infty
\end{aligned}$$

Bien que le nombre de contraintes du programme (23.2) est exponentiel (en la taille des entrées, n et m), il est possible de le résoudre à l'aide de l'algorithme des ellipsoïdes. L'exercice 23.1 proposera cependant un programme équivalent de taille polynomiale.

Aux chapitres 14 et 19, nous avons vu qu'il existait des relaxations linéaires pour certains problèmes **NP**-difficiles, tels que la couverture par sommets ou la coupe multiséparatrice, pour lesquels il existe toujours une solution optimale demi-entière. Arrondir les 1/2 à 1 dans ces solutions, donne alors une 2-approximation. La relaxation (23.2) a-t-elle cette propriété remarquable ? Le lemme suivant démontre que non.

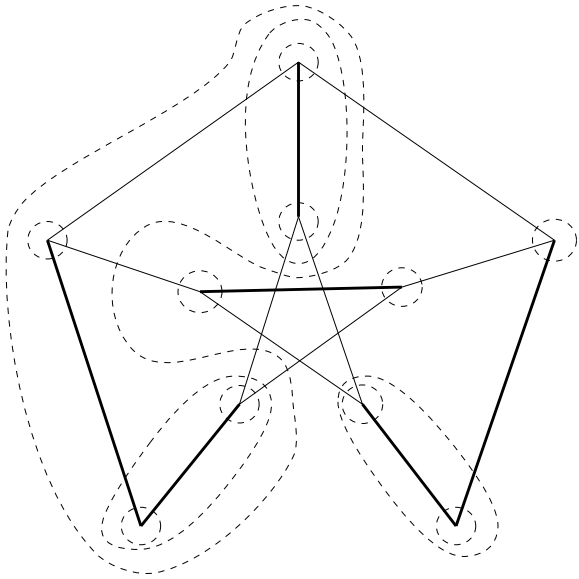
Lemme 23.2 *Considérons le graphe de Petersen (voir section 1.2) avec des coûts unitaires sur les arêtes et une requête de connexité unitaire pour chaque paire de sommets. Aucune solution optimale de la relaxation (23.2) n'est demi-entière pour cette instance.*

Preuve : Étudions la solution fractionnaire $x_e = 1/3$ pour chaque arête e. Comme le graphe de Peterson est 3-arête connexe (en fait, il est 3-sommet connexe également), c'est une solution réalisable. Le coût de cette solution est 5. Dans toute solution réalisable, la somme des variables sur les arêtes incidentes à tout sommet doit être supérieure à 1 pour garantir la connexité aux autres sommets. Comme ce graphe compte 10 sommets, le coût de toute solution réalisable est supérieur à 5. La solution ci-dessus est donc optimale.

Le coût de toute solution $\boldsymbol{x}$ ayant une arête e avec $x_e = 1$, est strictement supérieur à 5, car il faut des arêtes supplémentaires pour connecter les extrémités de e au reste du graphe. Ainsi, une solution demi-entière de coût optimal 5, serait nécessairement constituée d'un cycle hamiltonien, où chaque

arête est sélectionnée à moitié. Comme le graphe de Petersen n'a pas de cycle hamiltonien, il n'existe pas de solution optimale demi-entière. □

Rappelons qu'une solution d'un programme linéaire est *extrémale*[4] (ou encore, est un *sommet*, ou *basique réalisable*), si elle est réalisable et n'est combinaison linéaire convexe d'aucune paire de solutions réalisables distinctes. La solution $x_e = 1/3$ pour toute arête e, n'est pas extrémale. Une solution extrémale optimale est illustrée ci-après ; les arêtes en gras sont sélectionnées à moitié, les arêtes fines au quart, et les arêtes non représentées ne sont pas sélectionnées.



Le groupe des isomorphismes du graphe de Petersen est arête-transitif, et on obtient ainsi 15 solutions extrémales optimales distinctes ; la solution $x_e = 1/3$ pour toute arête e, est la moyenne de ces solutions.

Remarquons que bien qu'aucune solution extrémale ne soit demi-entière, certaines arêtes sont sélectionnées à moitié. Nous allons démontrer que cette propriété est vraie pour toute solution extrémale du programme (23.2). Nous obtiendrons alors une 2-approximation en arrondissant itérativement ces arêtes à 1. Notons H l'ensemble des arêtes sélectionnées par l'algorithme jusqu'à présent. La contrainte résiduelle de la coupe $(S, \overline{S})$ vaut $f'(S) = f(S) - |\delta_H(S)|$, où $\delta_H(S)$ est l'ensemble des arêtes de H traversant la coupe $(S, \overline{S})$. En général, les *contraintes de coupe résiduelles*, f', ne sont associées à aucun ensemble de requêtes de connexité. Les définitions suivantes seront utiles pour les caractériser.

[4] *Extreme point solution, vertex solution,* ou *basic feasible solution*, en anglais.

Nous dirons qu'une fonction $f : 2^V \to \mathbf{N}$ est *sous-modulaire*[5] si $f(V) = 0$, et si pour tous ensembles $A, B \subseteq V$, les deux inégalités suivantes sont vérifiées :

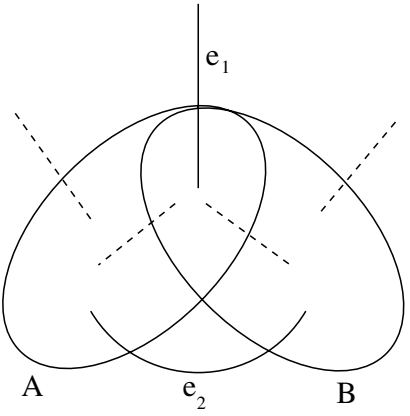
1. $f(A) + f(B) \geqslant f(A \cap B) + f(A \cup B)$
2. $f(A) + f(B) \geqslant f(A \setminus B) + f(B \setminus A)$.

Remarque 23.3 La sous-modularité n'est parfois définie que par la première inégalité. Nous aurons ici besoin de la définition plus forte donnée ci-dessus.

Nous dirons que deux sous-ensembles A et B de V *se croisent*[6] si chacun des ensembles $A \setminus B$, $B \setminus A$, et $A \cap B$ est non vide. Si A et B ne se croisent pas, alors soit ils sont disjoints, soit l'un contient l'autre.

Lemme 23.4 *Pour tout graphe G construit sur les sommets de V, la fonction $|\delta_G(.)|$ est sous-modulaire.*

Preuve : Si les ensembles A et B ne se croisent pas, les deux inégalités de sous-modularité sont clairement vérifiées. Dans le cas contraire, les arêtes ayant une extrémité dans $A \cap B$ et l'autre dans $\overline{A \cup B}$ (l'arête e_1 dans la figure ci-dessous) participe à $\delta(A)$ et à $\delta(B)$ mais pas à $\delta(A \setminus B)$, ni à $\delta(B \setminus A)$. De même, l'arête e_2 ci-dessous ne participe pas à $\delta(A \cap B)$, ni à $\delta(A \cup B)$. Le reste des arêtes participe de la même façon aux deux membres des deux inégalités. □



Nous dirons qu'une fonction $f : 2^V \to \mathbf{Z}$ est *faiblement super-modulaire*[7] si $f(V) = 0$ et si pour tous ensembles $A, B \subseteq V$, au moins l'une des deux inégalités suivantes est vraie :

1. $f(A) + f(B) \leqslant f(A \setminus B) + f(B \setminus A)$

[5] *Submodular function*, en anglais.
[6] *Crossing sets*, en anglais.
[7] *Weakly supermodular function*, en anglais.

2. $f(A) + f(B) \leqslant f(A \cap B) + f(A \cup B)$.

On voit facilement que les contraintes de coupe originales sont faiblement super-modulaires ; le lemme 23.5 démontre qu'il en va de même pour les contraintes de coupe résiduelles.

Lemme 23.5 *Considérons un sous-graphe H de G. Si $f : 2^{V(G)} \to \mathbf{N}$ est faiblement super-modulaire, alors les contraintes de coupe résiduelles f' le sont également.*

Preuve : Supposons que $f(A) + f(B) \leqslant f(A \smallsetminus B) + f(B \smallsetminus A)$; la preuve est identique pour l'autre cas. D'après le lemme 23.4, $|\delta_H(A)| + |\delta_H(B)| \geqslant |\delta_H(A \smallsetminus B)| + |\delta_H(B \smallsetminus A)|$. En soustrayant les deux inégalités, on obtient que $f'(A) + f'(B) \leqslant f'(A \smallsetminus B) + f'(B \smallsetminus A)$. □

Nous pouvons maintenant formuler la propriété polyédrale centrale requise pour notre 2-approximation. Cette propriété sera démontrée dans les sections suivantes.

Théorème 23.6 *Quelle que soit la fonction faiblement super-modulaire f, toute solution extrémale $\boldsymbol{x}$ du programme (23.2) sélectionne au moins une arête e au moins à moitié, c'est-à-dire $x_e \geqslant 1/2$.*

23.2 La technique de l'arrondi répété

Dans cette section, nous construisons un algorithme d'arrondi répété pour le problème du réseau de Steiner, fondé sur le théorème 23.6.

Algorithme 23.7 (Réseau de Steiner)

1. **Initialisation :** $H \leftarrow \varnothing$, $f' \leftarrow f$.
2. Tant que $f' \not\equiv \mathbf{0}$, faire :

   Trouver une solution extrémale $\boldsymbol{x}$ au programme (23.2) muni des contraintes de coupe f'.

   Pour toute arête e telle que $x_e \geqslant 1/2$, sélectionner $\lceil x_e \rceil$ copies de e dans H, et soustraire $\lceil x_e \rceil$ à u_e.

   Mise-à-jour de f' : pour tout $S \subseteq V$, $f'(S) \leftarrow f(S) - |\delta_H(S)|$.
3. Renvoyer H.

Le facteur d'approximation de l'algorithme ci-dessus est 2 quelle que soit la fonction faiblement super-modulaire f. Mais, pour obtenir un temps d'exécution polynomial, nous devons trouver rapidement une solution extrémale du programme (23.2). Or, nous ne savons pas comment faire

pour une fonction faiblement super-modulaire arbitraire f. Cependant, si la fonction f originale est construite à partir des requêtes de connexité, alors l'existence d'un oracle séparateur polynomial assure que chaque itération s'exécute en un temps polynomial.

Pour la première itération, l'oracle séparateur utilise une sous-routine de flot maximum. Nous associons à toute solution $\boldsymbol{x}$, un graphe de sommets V, où chaque arête e a une capacité x_e. Nous testons si pour toute paire de sommets $u, v \in V$, le graphe admet un flot supérieur à $r(u,v)$ de u à v. Si ce n'est pas le cas, nous obtenons une coupe violée, c'est-à-dire une coupe $(S, \overline{S})$ telle que $\delta_{\boldsymbol{x}}(S) < f(S)$, avec :

$$\delta_{\boldsymbol{x}}(S) = \sum_{e\,:\,e \in \delta(S)} x_e.$$

Notons f' les contraintes de coupe résiduelles pour les itérations suivantes. Nous associons à toute solution $\boldsymbol{x'}$ du programme (23.2) défini par f', la solution $\boldsymbol{x}$ suivante : pour toute arête e, $x_e = x'_e + e_H$, où e_H désigne le nombre de copies de l'arête e dans H. Le lemme suivant démontre que l'oracle séparateur pour la fonction de contraintes originale f donne un oracle séparateur pour f'. De plus, ce lemme démontre qu'il est inutile de mettre à jour f' explicitement à chaque itération.

Lemme 23.8 *Une coupe $(S, \overline{S})$ est violée par une solution $\boldsymbol{x'}$ sous les contraintes de coupe f' ssi elle est violée par la solution $\boldsymbol{x}$ sous les contraintes de coupe f.*

Preuve : Remarquons que $\delta_{\boldsymbol{x}}(S) = \delta_{\boldsymbol{x'}}(S) + |\delta_H(S)|$. Comme $f(S) = f'(S) + |\delta_H(S)|$, $\delta_{\boldsymbol{x}}(S) \geqslant f(S)$ ssi $\delta_{\boldsymbol{x'}}(S) \geqslant f'(S)$. □

Le lemme 23.8 implique qu'une solution $\boldsymbol{x'}$ est réalisable pour les contraintes de coupe f' ssi $\boldsymbol{x}$ est réalisable pour f. Le théorème 23.6, prouvé par les sections suivantes, nous permet de conclure que l'algorithme 23.7 est bien une 2-approximation.

Théorème 23.9 *L'algorithme 23.7 est une 2-approximation pour le problème du réseau de Steiner.*

Preuve : Nous allons démontrer, par récurrence sur le nombre d'itérations effectuées par l'algorithme à partir des contraintes de coupes faiblement super-modulaires f, que le coût de la solution entière générée est inférieur au double du coût de la solution fractionnaire optimale. Comme ce dernier minore le coût d'une solution entière optimale, le théorème s'ensuivra.

Si f nécessite une unique itération, le théorème est immédiat, puisque l'algorithme n'arrondit que les arêtes e telles que $x_e \geqslant 1/2$.

Supposons à présent que $\boldsymbol{x}$ est la solution extrémale optimale obtenue par la première itération. Construisons $\hat{\boldsymbol{x}}$ à partir de $\boldsymbol{x}$ en annulant toutes les coordonnées de $\boldsymbol{x}$ strictement inférieures à $1/2$. D'après le théorème 23.6, $\hat{\boldsymbol{x}} \neq 0$.

Notons H l'ensemble des arêtes sélectionnées lors de la première itération. Comme H est obtenu en arrondissant vers le haut toutes les coordonnées non nulles de $\hat{\boldsymbol{x}}$ et que ces coordonnées sont $\geqslant 1/2$, $\text{coût}(H) \leqslant 2 \cdot \text{coût}(\hat{\boldsymbol{x}})$.

Soient f' les contraintes de coupe résiduelles après la première itération, et H' l'ensemble des arêtes sélectionnées par les itérations suivantes pour satisfaire f'. La clé est d'observer que $\boldsymbol{x} - \hat{\boldsymbol{x}}$ est une solution réalisable pour f', et donc que $\text{coût}(H') \leqslant 2 \cdot \text{coût}(\boldsymbol{x} - \hat{\boldsymbol{x}})$, par hypothèse de récurrence. Notons $H + H'$ l'union des multi-ensembles d'arêtes H et H'. $H + H'$ satisfait clairement f. Ainsi,

$$\begin{aligned}\text{coût}(H + H') &\leqslant \text{coût}(H) + \text{coût}(H') \\ &\leqslant 2 \cdot \text{coût}(\hat{\boldsymbol{x}}) + 2 \cdot \text{coût}(\boldsymbol{x} - \hat{\boldsymbol{x}}) \leqslant 2 \cdot \text{coût}(\boldsymbol{x}).\end{aligned}$$ □

Corollaire 23.10 *Le saut intégral du programme (23.2) est inférieur à 2.*

Remarquons que les algorithmes à base d'arrondi décrits précédemment résolvaient le programme relâché une fois pour toute et effectuaient les arrondis sur la base de cette unique solution. Ces algorithmes n'exploitent pas totalement la puissance de l'arrondi – une fois qu'une partie de la solution a été arrondie, le reste fractionnaire de la solution n'est peut-être pas le meilleur choix pour poursuivre le processus d'arrondi. Il vaut peut-être mieux imposer les valeurs arrondies calculées aux variables arrondies, et recalculer une nouvelle solution fractionnaire pour le reste, comme c'est le cas ici. Cette technique est appelée *arrondi répété*.[8]

Exemple 23.11 L'instance critique proposée pour l'arbre de Steiner métrique (exemple 3.4) est également critique pour cet algorithme. Remarquons qu'après avoir sélectionné un sous-ensemble d'arêtes du cycle, toute solution extrémale optimale du programme résultant sélectionne toutes les arêtes restantes du cycle à moitié chacune. L'algorithme trouve donc une solution de coût $2(n-1)$, alors que le coût d'une solution optimale est n. □

23.3 Caractérisation des solutions extrémales

D'après les propriétés combinatoires des polyèdres, une solution réalisable d'un système d'inégalités dans $\mathbf{R}^m$ est extrémale ssi m inégalités linéairement indépendantes du système sont des égalités pour cette solution. Les solutions extrémales du programme (23.2) vérifient une propriété supplémentaire qui donnera le théorème 23.6.

Supposons que la fonction de contraintes de coupe f du programme (23.2) est une fonction faiblement super-modulaire arbitraire. Étant donné une solution $\boldsymbol{x}$ de ce programme linéaire, nous dirons qu'une inégalité est *exacte*

[8] *Iterated rounding*, en anglais.

si c'est une égalité pour $\boldsymbol{x}$. Si cette inégalité correspond à la contrainte de la coupe $(S, \bar{S})$, nous dirons que l'*ensemble S est exact*. Simplifions la solution considérée. Sans perte de généralité, nous pouvons ôter du graphe toutes les arêtes e telles que $x_e = 0$; et si $x_e \geqslant 1$, $\lfloor x_e \rfloor$ copies de l'arête e sont sélectionnées et nous pouvons simplifier les contraintes de coupe en conséquence. Nous pouvons donc supposer sans perte de généralité que pour toute solution extrémale $\boldsymbol{x}$, $(\forall e)\ 0 < x_e < 1$. Ainsi, à toute inégalité exacte correspond un ensemble exact. Notons m le nombre d'arêtes de G.

Nous dirons qu'une collection $\mathcal{L}$ de sous-ensembles de V est une *famille laminaire*[9] si aucune paire d'ensembles de la collection n'est croisée. Nous associons un vecteur de $\mathbf{R}^m$ à chaque inégalité correspondant à un ensemble S : ce vecteur vaut 1 sur chaque arête $e \in \delta_G(S)$, et 0 ailleurs. Ce vecteur, noté $\mathcal{A}_S$, est appelé *vecteur d'incidence* de S.

Théorème 23.12 *Pour chaque solution extrémale du programme (23.2), il existe une collection de m ensembles exacts telle que :*
- *leurs vecteurs d'incidence forment une famille libre, et*
- *cette collection d'ensembles est laminaire.*

Exemple 23.13 La solution extrémale pour le graphe de Petersen attribue des valeurs non nulles à 14 des 15 arêtes. D'après le théorème 23.12, il existe 14 ensembles exacts dont les vecteurs d'incidence sont linéairement indépendants. Ils sont marqués sur la figure page 241. □

Considérons une solution extrémale $\boldsymbol{x}$ du programme (23.2). Soit $\mathcal{L}$ une famille laminaire d'ensembles exacts dont les vecteurs d'incidence sont linéairement indépendants. Notons $\mathrm{vec}(\mathcal{L})$ le sous-espace vectoriel engendré par $\{\mathcal{A}_S : S \in \mathcal{L}\}$. Comme $\boldsymbol{x}$ est une solution extrémale, la dimension de la collection des ensembles exacts est m. Nous allons démontrer que si $\dim(\mathrm{vec}(\mathcal{L})) < m$, alors il existe un ensemble exact S, tel que si on l'ajoute à $\mathcal{L}$, $\mathcal{L}$ reste laminaire et la dimension de $\mathrm{vec}(\mathcal{L})$ augmente. Itérer ce procédé permet d'obtenir les m ensembles exacts du théorème 23.12.

Commençons par étudier les propriétés des ensembles exacts croisés.

Lemme 23.14 *Pour toute paire d'ensembles exacts croisés A et B, l'un des deux faits suivants est vrai :*
- *$A \setminus B$ et $B \setminus A$ sont tous deux exacts et $\mathcal{A}_A + \mathcal{A}_B = \mathcal{A}_{A \setminus B} + \mathcal{A}_{B \setminus A}$,*
- *$A \cup B$ et $A \cap B$ sont tous deux exacts et $\mathcal{A}_A + \mathcal{A}_B = \mathcal{A}_{A \cup B} + \mathcal{A}_{A \cap B}$.*

Preuve : Comme f est faiblement super-modulaire, soit $f(A) + f(B) \leqslant f(A \setminus B) + f(B \setminus A)$, ou bien $f(A) + f(B) \leqslant f(A \cup B) + f(A \cap B)$. Supposons que la première inégalité soit vraie ; la preuve est similaire pour la seconde. Comme A et B sont exacts, nous avons :

$$\delta_{\boldsymbol{x}}(A) + \delta_{\boldsymbol{x}}(B) = f(A) + f(B).$$

[9] *Laminar family*, en anglais.

Comme les contraintes associées à $A \smallsetminus B$ et $B \smallsetminus A$ sont respectées,

$$\delta_{\boldsymbol{x}}(A \smallsetminus B) + \delta_{\boldsymbol{x}}(B \smallsetminus A) \geqslant f(A \smallsetminus B) + f(B \smallsetminus A).$$

Ainsi,

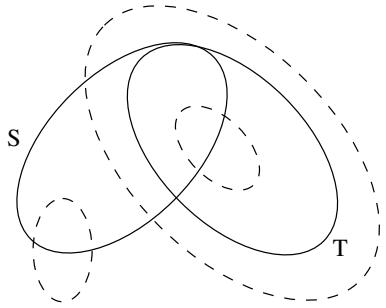
$$\delta_{\boldsymbol{x}}(A) + \delta_{\boldsymbol{x}}(B) \leqslant \delta_{\boldsymbol{x}}(A \smallsetminus B) + \delta_{\boldsymbol{x}}(B \smallsetminus A).$$

De même qu'au lemme 23.4 (qui établit que la fonction $|\delta_G(.)|$ est sous-modulaire), les arêtes ayant une extrémité dans $A \cup B$ et l'autre dans $A \cap B$ ne contribue qu'au membre de gauche de cette inégalité. Le reste des arêtes contribue identiquement aux deux membres. Cette inégalité est donc en fait une égalité. De plus, comme $x_e > 0$ pour toute arête e, aucune arête de G n'a une extrémité dans $A \cup B$ et l'autre dans $A \cap B$. Ainsi, $\mathcal{A}_A + \mathcal{A}_B = \mathcal{A}_{A \smallsetminus B} + \mathcal{A}_{B \smallsetminus A}$. □

Nous définissons le *nombre de croisement*[10] d'un ensemble $S \subseteq V$, comme le nombre d'ensembles de $\mathcal{L}$ qui croisent S.

Lemme 23.15 *Pour tout ensemble S qui croise un ensemble $T \in \mathcal{L}$, les nombres de croisement des ensembles $S \smallsetminus T$, $T \smallsetminus S$, $S \cup T$ et $S \cap T$ sont tous strictement inférieurs à celui de S.*

Preuve : Rappelons que $\mathcal{L}$ est une famille laminaire ; il suffit donc de compter les ensembles T' qui ne croisent pas T. La figure ci-dessous illustre les trois possibilités pour un ensemble $T' \in \mathcal{L}$ de croiser un des quatre ensembles sans croiser T (T' est représenté en pointillés). Dans tous les cas, T' croise S également. Enfin, T croise S mais aucun des quatre ensembles $S \smallsetminus T$, $T \smallsetminus S$, $S \cup T$ et $S \cap T$. □



Lemme 23.16 *Soit S un ensemble exact tel que $\mathcal{A}_S \notin \text{vec}(\mathcal{L})$, et tel que S croise un ensemble de $\mathcal{L}$. Alors, il existe un ensemble exact S' ayant un nombre de croisement strictement inférieur à celui de S et tel que $\mathcal{A}_{S'} \notin \text{vec}(\mathcal{L})$.*

[10] *Crossing number*, en anglais.

Preuve : Soit $T \in \mathcal{L}$ croisant S. Supposons que la première possibilité du lemme 23.14 soit vraie ; la preuve est similaire pour la seconde. Alors, $S \setminus T$ et $T \setminus S$ sont tous deux des ensembles exacts et $\mathcal{A}_S + \mathcal{A}_T = \mathcal{A}_{S \setminus T} + \mathcal{A}_{T \setminus S}$. Comme $\mathcal{A}_S \notin \text{vec}(\mathcal{L})$, cette identité linéaire implique que $\mathcal{A}_{S \setminus T}$ et $\mathcal{A}_{T \setminus S}$ ne peuvent pas appartenir tous les deux à $\text{vec}(\mathcal{L})$. D'après le lemme 23.15, les nombres de croisement de $S \setminus T$ et $T \setminus S$ sont tous deux strictement inférieurs à celui de S, CQFD. □

Corollaire 23.17 *Si* $\text{vec}(\mathcal{L}) \neq \mathbf{R}^m$*, alors il existe un ensemble exact* S *tel que* $\mathcal{A}_S \notin \text{vec}(\mathcal{L})$ *et* $\mathcal{L} \cup \{S\}$ *est une famille laminaire.*

Le corollaire 23.17 implique que, si $\mathcal{L}$ est une famille laminaire maximale d'ensembles exacts dont les vecteurs d'incidence sont linéairement indépendants, alors $|\mathcal{L}| = m$. Ceci conclut la démonstration du théorème 23.12.

23.4 Un argument de dénombrement

La caractérisation des solutions extrémales du théorème 23.12 conduit au théorème 23.6 par dénombrement. Considérons une solution extrémale $\boldsymbol{x}$ et la collection d'ensembles exacts $\mathcal{L}$ construite au théorème 23.12. Le nombre d'ensembles dans $\mathcal{L}$ est m, le nombre d'arêtes de G. Raisonnons par l'absurde et supposons que $x_e < 1/2$ pour toute arête e. Nous allons démontrer que ceci implique que G a strictement plus de m arêtes.

Comme $\mathcal{L}$ est une famille laminaire, nous pouvons la voir comme une forêt dont les éléments sont ordonnés par inclusion. Nous dirons qu'un ensemble $S \in \mathcal{L}$ est un *ensemble racine*, si S n'est inclus dans aucun autre ensemble de $\mathcal{L}$. Si S n'est pas un ensemble racine, nous dirons que T est le *père* de S si T est un ensemble minimal de $\mathcal{L}$ contenant S. Comme $\mathcal{L}$ est laminaire, un tel T est unique. Nous dirons alors que S est un *fils* de T. La relation de *descendance* est la fermeture reflexive et transitive de la relation « être le fils de ». Les *feuilles* sont les ensembles sans enfant. $\mathcal{L}$ est donc une forêt d'arbres, où chaque arbre est enraciné sur un ensemble racine. Nous appellerons *sous-arbre enraciné en* S l'ensemble des descendants de S.

Une arête e est dite *incidente* à un ensemble S si $e \in \delta_G(S)$. Le *degré* de S est la quantité $|\delta_G(S)|$. Nous dirons que S *contrôle* l'extrémité v d'une arête uv si S est le plus petit ensemble de $\mathcal{L}$ qui contient v. De même, le sous-arbre enraciné en S *contrôle* l'extrémité v d'une arête uv si un des descendants de S contrôle v.

Puisque G a m arêtes, G compte $2m$ extrémités. Nous allons démontrer, sous l'hypothèse $(\forall e)\ x_e < 1/2$, que pour tout ensemble S, nous pouvons distribuer les extrémités contrôlées par le sous-arbre enraciné en S de sorte que S reçoit au moins trois extrémités, et que chacun de ses descendants propres en reçoit deux. L'application de cette procédure à chacune des racines de la

forêt démontrera que le nombre d'extrémités dans le graphe est strictement supérieur à $2m$, ce qui est une contradiction.

Par hypothèse, $(\forall e)\ 0 < x_e < 1/2$. Posons $y_e = 1/2 - x_e$, le *complément à la moitié* de e, pour chaque arête e. Clairement, $0 < y_e < 1/2$. Définissons la *cocontrainte* de $S \in \mathcal{L}$, comme la quantité

$$\text{coreq}(S) = \sum_{e \in \delta(S)} y_e = \frac{1}{2}|\delta_G(S)| - f(S).$$

Clairement, $0 < \text{coreq}(S) < |\delta_G(S)|/2$. Comme $|\delta_G(S)|$ et $f(S)$ sont tous deux entiers, $\text{coreq}(S)$ est demi-entier. Nous dirons que $\text{coreq}(S)$ est *semi-entier* s'il n'est pas entier, c'est-à-dire si $\text{coreq}(S) \in \{1/2, 3/2, 5/2, \ldots\}$. Comme $f(S)$ est entier, $\text{coreq}(S)$ est semi-entier ssi $|\delta_G(S)|$ est impair.

Les ensembles de cocontrainte $1/2$ joue un rôle particulier dans cette preuve. Le lemme suivant permettra de caractériser certains ensembles de cocontrainte $1/2$.

Lemme 23.18 *Soit S un ensemble ayant α fils et contrôlant β extrémités, avec $\alpha + \beta = 3$. Si les cocontraintes des fils de S sont toutes $1/2$, alors* $\text{coreq}(S) = 1/2$.

Preuve : Comme les cocontraintes des fils de S (s'il en existe) sont toutes $1/2$, chaque fils est de degré impair. Comme $\alpha + \beta = 3$, S est également de degré impair (voir exercice 23.3). Ainsi, la cocontrainte de S est semi-entière. Il suffit maintenant de montrer que $\text{coreq}(S) < 3/2$, pour conclure. Or,

$$\text{coreq}(S) = \sum_{e \in \delta(S)} y_e \leqslant \sum_{S' \text{ fils de } S} \text{coreq}(S') + \sum_{e\,:\,|e \cap S|=1} y_e,$$

Comme $y_e < 1/2$, si $\beta > 0$ alors $\text{coreq}(S) < 3/2$. Si $\beta = 0$, toutes les arêtes incidentes aux fils de S ne peuvent être incidentes à S, sinon les vecteurs d'incidence de ces quatre ensembles seraient liés linéairement. Ainsi,

$$\text{coreq}(S) < \sum_{S' \text{ fils de } S} \text{coreq}(S') = 3/2.$$

□

Les deux lemmes suivants minorent le nombre d'extrémités contrôlées.

Lemme 23.19 *Si un ensemble S a un fils unique, alors S contrôle au moins deux extrémités.*

Preuve : Soit S' le fils de S. Si S n'a pas d'extrémité qui lui est incidente, les arêtes incidentes à S et à S' sont les mêmes. Mais alors $\mathcal{A}_S = \mathcal{A}_{S'}$, contradiction. Il est également impossible que S ne contrôle qu'une unique

extrémité, car sinon, la différence de $\delta_{\boldsymbol{x}}(S)$ et $\delta_{\boldsymbol{x}}(S')$ serait fractionnaire, contredisant que ces deux ensembles sont exacts et de contrainte entière. CQFD. □

Lemme 23.20 *Si un ensemble S a deux fils, dont l'un a pour cocontrainte $1/2$, alors S contrôle au moins une extrémité.*

Preuve : Soient S' et S'' les deux fils de S, avec $\text{coreq}(S') = 1/2$. Supposons que S ne contrôle aucune extrémité. Puisque les trois vecteurs $\mathcal{A}_S$, $\mathcal{A}_{S'}$, et $\mathcal{A}_{S''}$ sont linéairement indépendants, les arêtes incidentes à S' ne peuvent être toutes incidentes à S ou toutes incidentes à S''. Notons a la somme des y_e sur toutes les arêtes e incidentes à S' et à S, et b la somme des y_e sur toutes les arêtes e incidentes à S' et à S''. Nous avons $a > 0$, $b > 0$, et $a + b = \text{coreq}(S) = 1/2$.

Puisque la cocontrainte de S' est semi-entière, le degré de S' est impair. Par conséquent, les parités des degrés de S et S'' sont différentes, et les cocontraintes de ces deux ensembles sont différentes. Or, $\text{coreq}(S) = \text{coreq}(S'') + a - b$. Ainsi, $\text{coreq}(S) - \text{coreq}(S'') = a - b$. Mais, $-1/2 < a - b < 1/2$. Ainsi, S et S'' ont la même cocontrainte, contradiction. CQFD. □

Lemme 23.21 *Soit T un arbre enraciné sur un ensemble S. Sous l'hypothèse que $(\forall e)\ x_e < 1/2$, on peut distribuer les extrémités contrôlées par T sur les descendants de S, de sorte que S reçoit au moins trois extrémités, et que chacun des descendants propres de S en reçoit au moins deux. De plus, si $\text{coreq}(S) \neq 1/2$, alors S reçoit au moins quatre extrémités.*

Preuve : Procédons par récurrence sur la hauteur de l'arbre T. Commençons par le cas où S est une feuille. Le degré de S est alors supérieur à 3, sinon toute arête e incidente à S vérifierait $x_e \geqslant 1/2$. Si le degré de S est exactement 3, $\text{coreq}(S)$ est semi-entier. Comme $\text{coreq}(S) < |\delta_G(S)|/2 = 3/2$, la cocontrainte de S est donc $1/2$. Or S est une feuille, et contrôle donc une extrémité de chacune de ses arêtes incidentes. S contrôle donc bien le nombre requis d'extrémités.

Disons qu'un ensemble a un *excédent* de 1 si 3 extrémités lui ont été assignées et un excédent de 2 si 4 lui ont été assignées. Considérons maintenant le cas où S n'est pas une feuille. La preuve consiste à démontrer qu'en reportant les excédents des fils de S tout en tenant compte des extrémités contrôlées par S lui-même, on peut attribuer à S le nombre requis d'extrémités. Quatre cas sont à considérer :

1. Si S a au moins 4 fils, il suffit de lui attribuer les excédents de ses fils pour garantir qu'au moins 4 extrémités lui sont attribuées.
2. Supposons que S a 3 fils. Si l'un d'eux a un excédent de 2, ou si S contrôle une extrémité, on peut attribuer 4 extrémités à S. Sinon, les cocontraintes de chacun des enfants de S valent toutes $1/2$, et d'après le lemme 23.18,

coreq$(S) = 1/2$ également. Ainsi, attribuer à S les excédents de ses fils suffit.

3. Supposons que S a deux fils. Si chacun de ses fils a un excédent de 2, on peut attribuer 4 extrémités à S. Si l'un d'eux a un excès de 1, le lemme 23.20 garantit que S contrôle au moins une extrémité. Si les excédents des deux fils valent 1 et que S contrôle exactement une extrémité, alors on peut aussi attribuer 3 extrémités à S, et cela suffit d'après le lemme 23.18. Dans les autres cas, on peut attribuer 4 extrémités à S.
4. Si S a un fils unique S', alors d'après le lemme 23.19, S contrôle au moins 2 extrémités. Si S contrôle exactement deux extrémités et si l'excédent de S' est exactement 1, alors on peut attribuer 3 extrémités à S ; comme d'après le lemme 23.18, coreq$(S) = 1/2$, cela suffit. Dans tous les autres cas, on peut attribuer 4 extrémités à S.

□

23.5 Exercices

23.1 Donnez une relaxation linéaire pour le problème du réseau de Steiner, ayant un nombre polynomial de contraintes et de variables.
Indication : Sélectionnez un ensemble d'arêtes de coût minimum pour router $\binom{n}{2}$ flots indépendants, un flot par paire de sommets. Chaque flot devra être supérieur à la contrainte de connexité de la paire associée, et chaque arête sera sélectionnée dans une proportion qui borne la valeur de chacun des flots qui la traverse.

23.2 Démontrez que toute fonction $f : 2^V \to \mathbf{N}$ vérifiant les conditions suivantes est sous-modulaire : $f(V) = 0$, f est symétrique (c'est-à-dire $(\forall A \subseteq V)\ f(A) = f(V \setminus A)$), et pour toute paire d'ensembles $A, B \subseteq V$, $f(A) + f(B) \geqslant f(A \cap B) + f(A \cup B)$.

23.3 Démontrez que le degré de l'ensemble S du lemme 23.18 est impair.
Indication : Envisagez les trois cas suivants : S contrôle une extrémité v d'une arête uv 1) incidente à S, ou bien 2) incidente à un fils de S, ou bien 3) incidente à deux fils de S.

23.4 Montrez qu'il existe un ensemble de degré inférieur à 3 dans $\mathcal{L}$, et donc qu'il existe une arête e telle que $x_e \geqslant 1/3$. L'argument de dénombrement est alors beaucoup plus simple. Remarquez que ceci donne une 3-approximation (le dénombrement utilise le lemme 23.19).

Les deux exercices suivants développent sur la base du schéma primal-dual, une $2H_k$-approximation pour le problème du réseau de Steiner, où k

est la plus grande contrainte de connexité de l'instance. Pour simplifier, nous supposerons que les bornes u_e valent 1 sur toutes les arêtes e.

23.5 (Williamson, Goemans, Mihail, et Vazirani [267]) Nous dirons qu'une fonction $h : 2^V \to \{0,1\}$ est *infroissable*[11] si $h(V) = 0$, et si pour toute paire d'ensembles $A, B \subset V$, si $h(A) = h(B) = 1$ alors $h(A \smallsetminus B) = h(B \smallsetminus A) = 1$ ou $h(A \cap B) = h(A \cup B) = 1$. L'exercice 22.7 demandait une 2-approximation pour le programme en nombres entiers (22.1) dans le cas où f est une fonction propre. Dans cet exercice, nous allons étendre ce résultat au cas où f est infroissable. Nous devons modifier la dernière étape de l'algorithme 22.3 ; l'étape d'élimination sera exécutée de l'arrière vers l'avant. Comme précédemment, notons F la forêt des arêtes sélectionnées par l'algorithme. Nous dirons qu'un ensemble $A \subset V$ est *insatisfait* par F si $h(A) = 1$ et $\delta_F(A) = \varnothing$. Nous dirons qu'un ensemble insatisfait minimal est *actif*. Voici l'algorithme.

Algorithme 23.22 (Fonction infroissable)

1. **(Initialisation)** $F \leftarrow \varnothing$; pour tout $S \subseteq V$, $y_S \leftarrow 0$.
2. **(Augmentation des arêtes)**
   Tant qu'il existe un ensemble insatisfait, faire :
   Augmenter simultanément les y_S des ensembles actifs S, jusqu'à ce qu'une arête e soit saturée ;
   $F \leftarrow F \cup \{e\}$.
3. Notons $e_1, e_2, \ldots, e_l$ la liste des arêtes de F par ordre d'insertion.
4. **(Élimination en arrière)** Pour j décroissant de l à 1, faire :
   Si $F \smallsetminus \{e_j\}$ satisfait h, alors $F \leftarrow F \smallsetminus \{e_j\}$.
5. Renvoyer F.

Montrez que durant chaque itération, les ensembles actifs sont tous disjoints. En supposant qu'on puisse construire efficacement les ensembles actifs, démontrez que l'algorithme 23.22 trouve une solution primale de coût inférieur au double du dual, c'est-à-dire telle que

$$\sum_{e \in F} c_e \leqslant 2 \sum_S y_S.$$

Indication : On peut associer à chaque arête $e \in F$, un ensemble $A \subset V$ tel que $h(A) = 1$ et $\delta_F(A) = \{e\}$. Cet ensemble est appelé le *témoin*[12] (souvent aussi le *certificat*) de e. Nous appelons *famille de certificats* toute collection $\mathcal{C}$ contenant un certificat pour chaque $e \in F$. Ajoutons V à une telle famille $\mathcal{C}$.

[11] *Uncrossable function*, en anglais.
[12] *Witness*, en anglais.

Démontrez, à l'aide de l'infroissabilité, que $\mathcal{C}$ peut être construite laminaire et donc être vue comme un arbre. En conclure qu'à chaque itération, le degré moyen des ensembles actifs est inférieur à 2, comme au lemme 22.8.

23.6 Exhibez une instance pour laquelle il existe une fonction infroissable telle que si l'étape d'élimination en arrière est effectuée en avant, le facteur d'approximation de l'algorithme 23.22 tend vers l'infini.

23.7 (Goemans, Goldberg, Plotkin, Shmoys, Tardos et Williamson [109]) Nous allons résoudre le problème du réseau de Steiner en k phases de calcul, numérotées de 0 à $k-1$. Chaque phase calcule une forêt du graphe courant. La solution sera l'union de ces k forêts. Notons F_{p-1} l'ensemble des arêtes sélectionnées durant les phases de 0 à $p-1$. Au début de la p-ième phase, nous définissons le *défaut*[13] d'un ensemble $S \subset V$, comme la quantité $\max\{f(S) - |\delta_{F_{p-1}}(A)|, 0\}$. Les $p-1$ premières phases garantissent que le défaut de tout ensemble est inférieur à $k-p$. Durant la p-ième phase, nous définissons la fonction h par :

$$h(S) = \begin{cases} 1 \text{ si défaut}(S) = k-p \\ 0 \text{ sinon} \end{cases}$$

Montrez que h est infroissable. Montrez qu'il existe une implémentation de l'algorithme 23.22 en temps polynomial pour cette fonction infroissable, c'est-à-dire que les ensembles actifs se trouvent en temps polynomial. Notons F l'ensemble des arêtes sélectionnées par l'algorithme 23.22 à partir de $E \setminus F_{p-1}$, et $\boldsymbol{y}$ la solution duale construite durant l'exécution sur la fonction h. Donnez le programme dual de (23.2), et montrez qu'il existe une solution réalisable $\boldsymbol{d}$ de ce programme, telle que :

$$\sum_{e \in F} c_e \leqslant 2 \sum_S y_S \leqslant \frac{2}{k-p} g(\boldsymbol{d}),$$

où $g(\boldsymbol{d})$ est la valeur objectif de la solution duale $\boldsymbol{d}$. Déduisez-en le facteur d'approximation souhaité en sommant sur les k phases.
Indication : Utilisez un algorithme de flot maximum pour trouver les ensembles actifs. Le programme dual associe une variable z_e à chaque arête e. Obtenez une solution duale réalisable en posant $z_e = \sum_{S\,:\,e \in \delta(S)} y_S$, pour chaque arête $e \in F_{p-1}$.

23.8 Exhibez une famille de graphes montrant que le facteur d'approximation de l'algorithme de l'exercice 23.7 est exact à un facteur constant près.

Les définitions suivantes seront utilisées par les trois exercices suivants. Le théorème de Menger relie les objets étudiés au thème de ce chapitre, l'étude

[13] *Deficiency of a set*, en anglais.

de petits sous-graphes ayant un nombre requis de chemins disjoints (voir exercice 12.6). Nous dirons qu'un graphe non orienté est *k-sommet (k-arête) connexe*[14] s'il possède au moins $k+1$ sommets, et que le retrait de n'importe quel sous-ensemble de moins de $k-1$ sommets (arêtes) ne déconnecte pas le graphe. Un graphe orienté est dit *k-sommet (k-arc) connexe*[15] s'il a au moins $k+1$ sommets, et que le retrait de n'importe quel sous-ensemble de moins de $k-1$ sommets (arcs) ne déconnecte pas le graphe fortement.

23.9 (Cheriyan et Thurimella [45]) Cet exercice propose une $(1+2/k)$-approximation pour le problème suivant.

Problème 23.23 (Sous-graphe k-sommet connexe minimum)[16] Étant donné un entier positif k et un graphe non orienté $G=(V,E)$ k-sommet connexe, trouver un sous-ensemble d'arêtes de taille minimum $E' \subset E$ tel que le sous-graphe $G'=(V,E')$ soit k-sommet connexe.

Soit $G=(V,E)$ un graphe k-sommets connexe. Nous dirons qu'une arête $e \in E$ est *critique* si, suite à l'élimination de e, le graphe n'est plus k-sommet connexe. Un cycle simple C de G est dit *critique* si toutes les arêtes de C sont critiques. L'algorithme repose sur le théorème de Mader qui affirme que tout cycle critique de G admet un sommet de degré exactement k.

Algorithme 23.24 (Sous-graphe k-sommet connexe)

1. Trouver un sous-ensemble de taille minimum $M \subset E$ tel que $(\forall v \in V)\ \deg_M(v) \geqslant k-1$.
2. Trouver un sous-ensemble minimal F tel que $M \cup F$ est k-sommet connexe.
3. Renvoyer $G'=(V, M \cup F)$.

1. Donnez un algorithme polynomial pour l'étape 1 de l'algorithme 23.24. Remarquez que $|M| \leqslant \mathrm{OPT}$.
   Indication : Utilisez un algorithme de b-couplage[17] sur le complément de G. Étant donné un graphe non orienté $G=(V,E)$ et des bornes $b : V \to \mathbf{N}$ sur les sommets, le problème du b-couplage consiste à trouver un sous-ensemble $M \subseteq E$ de taille maximum tel que $(\forall v \in V)\ \deg_M(v) \leqslant b(v)$. Ce problème est dans **P**.
2. Proposez une implémentation efficace de l'étape 2 de l'algorithme 23.24.

[14] *k-vertex (k-edge) connected graph*, en anglais.
[15] Idem, *k-vertex (k-edge) connected graph*, en anglais.
[16] *Minimum k-vertex connected subgraph*, en anglais.
[17] *b-Matching*, en anglais.

3. Démontrez à l'aide du théorème de Mader que F est acyclique, et donc que $|F| \leqslant |V| - 1$. En conclure que le facteur d'approximation de l'algorithme 23.24 est $1 + 2/k$.
   Indication : Remarquez que OPT est minoré par $k|V|/2$.

23.10 (Cheriyan et Thurimella [45]) Étudions le problème de trouver un sous-graphe k-sommet connexe minimum dans un graphe orienté. Proposez un algorithme semblable à celui de l'exercice 23.9 réalisant une approximation de facteur $1 + 2/k$ pour ce problème. Utilisez les deux faits suivants :

1. Dans un graphe orienté, un *cycle alterné* C est une suite d'arcs distincts $(v_0 \to v_1), (v_1 \leftarrow v_2), (v_2 \to v_3), (v_3 \leftarrow v_4), \ldots, (v_{m-1} \to v_m), (v_m \leftarrow v_0)$ de longueur paire, où les sommets ne sont pas nécessairement distincts. Remarquez que les sommets de C ont alternativement deux arcs sortants et deux arcs entrants. Les sommets ayant deux arcs sortants (entrants) sont dits *C-sortant* (*C-entrant*). Le théorème de Mader montre que si G est un graphe k-sommet fortement connexe contenant un cycle alterné C, dont les arcs sont tous critiques, alors C contient soit un sommet C-sortant de degré sortant exactement k ou un sommet C-entrant de degré entrant k exactement.
2. Définissons le *graphe biparti associé* H d'un graphe orienté $G = (V, E)$, de la façon suivante. H a deux sommets v_- et v_+ pour chaque sommet $v \in V$, et une arête u_+v_- pour chaque arc $(u, v) \in E$. Alors, G admet un cycle alterné ssi son graphe biparti associé admet un cycle.

23.11 (Khuller et Vishkin [180], fondé sur Edmonds [79]) Cet exercice développe une 2-approximation pour le problème suivant.

Problème 23.25 (Sous-graphe k-arête connexe minimum)[18] Étant donné un graphe non orienté $G = (V, E)$, des poids $w : E \to \mathbf{Q}^+$, et un entier k, trouver un sous-graphe k-arête connexe de poids minimum de G.

1. Distinguons un sommet $r \in V$ arbitraire de G. Étudions le problème de trouver un sous-graphe de poids minimum G' de G, tel que pour tout sommet $v \in V$, il existe k chemins arêtes-disjoints de r à v dans G'. Démontrez que ce problème est le même que le problème 23.25, c'est-à-dire que toute solution de l'un est solution de l'autre.
2. Soit $G = (V, E)$ un graphe orienté avec des poids sur les arêtes et $r \in V$ un de ses sommets. Nous dirons qu'un ensemble d'arcs $E' \subseteq E$ est une *r-arborescence* si le degré entrant de tout sommet $u \neq r$ est 1 ; une r-arborescence est un arbre couvrant orienté depuis r. Nous appellerons *r-connexité* de G la quantité :

$$\max\{k : \forall v \in V,\ \exists k \text{ chemins arête-disjoints de } r \text{ à } v \text{ dans } G\}.$$

[18] *Minimum k-edge connected subgraph*, en anglais.

Edmonds a démontré que le nombre maximum de r-arborescences arcs-disjointes dans G est égal à la r-connexité de G. Déduisez-en que trouver un sous-graphe de G de poids minimum de r-connexité k revient exactement à trouver un sous-graphe de G de poids minimum qui a k r-arborescences arcs-disjointes.

3. Edmonds a montré qu'on peut partitionner les arcs d'un graphe orienté $G = (V, E)$ en k r-arborescences arcs-disjointes ssi, en ignorant les orientations des arcs, E peut être partionné en k arbres couvrants, et que le degré entrant de chaque sommet, autre que r, est exactement k. Déduisez-en qu'on peut trouver en temps polynomial un sous-graphe de G de poids minimum ayant k arborescences arcs-disjointes.

   Indication : Ce problème peut se formuler comme un problème d'intersection de matroïdes, les deux matroïdes étant un matroïde de partitions et l'union de k couches d'un matroïde graphique (qui est aussi un matroïde).

4. Soient $G = (V, E)$ un graphe non orienté avec des poids sur les arêtes, et $r \in V$ un de ses sommets. Notons $\mathrm{OPT}(G)$ le poids d'une solution optimale au problème 23.25 pour l'instance G. Construisons le graphe H en « bi-orientant » les arêtes de G, c'est-à-dire en remplaçant chaque arête $uv \in E$ par deux arcs $(u \to v)$ et $(v \to u)$, ayant chacun le même poids que uv. Notons $\mathrm{OPT}(H)$ le poids minimum d'un sous-graphe de H qui se décompose en k r-arborescences. Montrez que :

   $$\mathrm{OPT}(G) \leqslant \mathrm{OPT}(H) \leqslant 2 \cdot \mathrm{OPT}(G).$$

   Déduisez-en une 2-approximation pour le problème 23.25.

23.12 (Goemans et Bertsimas [108]) Le problème du réseau de Steiner métrique est le problème du réseau de Steiner restreint aux instances où G est un graphe complet avec une fonction de coût métrique sur les arêtes et une borne $u_e = \infty$ sur chaque arête e. C'est une généralisation du problème de l'arbre de Steiner métrique au cas de requêtes de connexité arbitraires. Pour tout $D \subseteq V$, notons $\mathrm{LP}_S(D)$ la relaxation linéaire (23.2) complétée par des contraintes d'égalités sur les sommets de D, comme suit.

$$\begin{aligned}
&\text{minimiser} && \sum_{e \in E} c_e x_e && && (23.4)\\
&\text{sous les contraintes} && \sum_{e\,:\,e \in \delta(S)} x_e \geqslant f(S), && S \subseteq V\\
& && \sum_{e\,:\,e \text{ incidente à } v} x_e = f(\{v\}), && v \in D\\
& && x_e \geqslant 0, && e \in E
\end{aligned}$$

Il se trouve que ces contraintes d'égalités sont redondantes pour le problème du réseau de Steiner métrique. Quel que soit $D \subseteq V$, une solution optimale de $\mathrm{LP}_S(D)$ est également une solution de $\mathrm{LP}_S(\varnothing)$. C'est la *propriété de parcimonie*. Nous dirons qu'un sommet v est *Steiner* s'il n'a pas de contrainte de connexité, c'est-à-dire si $(\forall u \in V)\ r(u,v) = 0$. Utilisez la propriété de parcimonie pour démontrer qu'il existe une solution fractionnaire optimale au problème de Steiner métrique dans laquelle aucune arête n'est incidente à un sommet Steiner.

23.13 Considérons le programme en nombres entiers suivant pour le problème du représentant de commerce (problème 3.5).

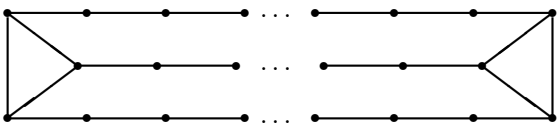
$$\begin{aligned}
&\text{minimiser} && \sum_{e\in E} c_e x_e && && (23.5)\\
&\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e = 2, && v \in V\\
& && \sum_{e\,:\,e\in\delta(S)} x_e \geqslant 2, && S \subset V\\
& && x_e \in \{0,1\}, && e \in E
\end{aligned}$$

Montrez que les solutions optimales de ce programme sont des tours optimaux pour le TSP. La relaxation linéaire de ce programme est appelée la relaxation par *élimination des sous-tours*[19] pour le TSP.

La suite de cet exercice traite du cas métrique du TSP et prouve que la solution construite par l'algorithme de Christofides (algorithme 3.10) est à un facteur 3/2 de la solution optimale de cette relaxation.

1. Exhibez une famille d'instances qui minore le saut intégral de la relaxation par 4/3 (asymptotiquement).

   Indication : Inspirez-vous du graphe suivant.



2. Distinguons un sommet v_1 arbitraire du graphe $G = (V, E)$. Nous appellerons 1-arbre de G tout arbre couvrant les sommets de $V \smallsetminus \{v_1\}$, complété par deux arêtes distinctes incidentes au sommet v_1. Le coût minimal d'un 1-arbre optimal minore clairement le coût d'un tour TSP optimal. Le programme (12.12) de l'exercice 12.10 est une relaxation exacte pour le MST. Utilisez-la pour obtenir une relaxation linéaire exacte pour le problème du 1-arbre minimum.

[19] *Subtour elimination relaxation for TSP*, en anglais.

3. (Held et Karp [131]) Démontrez que le coût minimum d'un 1-arbre minore le coût d'une solution optimale du programme par élimination des sous-tours.
   Indication : Comparez les programmes linéaires obtenus ci-dessus pour le 1-arbre minimum et la formulation équivalente suivante du programme par élimination des sous-tours (on entend par $e \subseteq S$, que e a ses deux extrémités dans S).

$$\begin{aligned} &\text{minimiser} && \sum_{e \in E} c_e x_e && && (23.6)\\ &\text{sous les contraintes} && \sum_{e\,:\,e \text{ incidente à } v} x_e = 2, && v \in V\\ & && \sum_{e\,:\,e \subseteq S} x_e \leqslant |S| - 1, && S \subseteq V\\ & && x_e \geqslant 0, && e \in E \end{aligned}$$

4. Déduisez de la propriété de parcimonie de l'exercice 23.12, que les contraintes d'égalité sur les sommets dans le programme par élimination des sous-tours, sont redondantes. Remarquez que le programme obtenu en éliminant ces contraintes est une relaxation linéaire du problème de trouver un sous-graphe 2-arête connexe de G de coût minimum.
5. Pour tout $D \subseteq V$, notons $\mathrm{LP}_T(D)$ le programme par élimination des sous-tours pour le sous-graphe induit G_D par D dans G. Notons $\mathrm{OPT}_f(\mathrm{LP}_T(D))$ le coût d'une solution optimale de $\mathrm{LP}_T(D)$. Démontrez la propriété de monotonie suivante :

   $$\mathrm{OPT}_f(\mathrm{LP}_T(D)) \leqslant \mathrm{OPT}_f(\mathrm{LP}_T(V)).$$

   Indication : Utilisez la relaxation sans les contraintes d'égalité.
6. Soit $D \subseteq V$ de taille paire. Montrez que le coût minimum d'un couplage parfait du sous-graphe de G induit par D est $\leqslant \frac{1}{2}\,\mathrm{OPT}_f(\mathrm{LP}_T(D))$.
   Indication : Utilisez le programme (12.9) de l'exercice 12.9 pour le couplage, et le programme (23.6) pour le TSP.
7. Démontrez que la solution pour le TSP métrique trouvée par l'algorithme 3.10 est à un facteur 3/2 du coût optimal d'une solution optimale du programme par élimination des sous-tours.

23.6 Notes

Le premier algorithme d'approximation pour ce problème, d'un facteur $2k$, où k est la plus grande requête de connexité de l'instance, est dû à Williamson,

Goemans, Mihail et Vazirani [267]. Cet algorithme était fondé sur le schéma primal-dual. Cette approche a été améliorée par Goemans, Goldberg, Plotkin, Shmoys, Tardos et Williamson [109], pour obtenir un facteur d'approximation de $2H_k$. Le résultat présenté dans ce chapitre est dû à Jain [145]. Cheriyan et Thurimella proposent dans [45] des résultats supplémentaires sur la recherche de petits sous-graphes d'un graphe vérifiant des contraintes de connexité, ainsi que les références au théorème de Mader. La relaxation par élimination des sous-tours pour le TSP est due à Dantzig, Ford et Fulkerson [63]. Le résultat de l'exercice 23.13 fut tout d'abord obtenu par Wolsey [270]. La preuve présentée ici est de Shmoys et Williamson [249].

24 Placement d'installations

Le problème du placement d'installations a occupé une place centrale en recherche opérationnelle depuis la fin des années 1960. Ce problème modélise la détermination des emplacements d'usines, d'entrepôts, d'écoles, ou d'hôpitaux, ainsi que le placement de serveurs proxy sur le web.

Nous présentons dans ce chapitre, une 3-approximation dérivée du schéma primal-dual, pour le cas où les coûts de connexion sont métriques. Cet algorithme diffère en deux points des algorithmes primal-dual précédents. Premièrement, les programmes primal et dual ont des coefficients négatifs et ne forment donc pas de paire couverture-empaquetage. Ensuite, nous allons relâcher les conditions primales des écarts complémentaires plutôt que les duales. Par ailleurs, nous raffinerons encore le principe de synchronisation introduit au chapitre 22 pour le schéma primal-dual, en faisant intervenir les dates des différent événements.

Problème 24.1 (Placement d'installations métrique et sans capacité)[1] Soit $G = ((F, C), E)$ un graphe biparti, où F est l'ensemble des *installations*[2] et C l'ensemble des *villes*. Notons f_i le coût de construction de l'installation i, et c_{ij} le coût de connexion de la ville j à l'installation i. Les coûts de connexion sont métriques (c'est-à-dire pour tout i, j, i', et j', $c_{ij} \leqslant c_{ij'} + c_{i'j'} + c_{i'j}$). Le problème est de trouver un sous-ensemble $I \subseteq F$ d'installations à construire, et une assignation $\phi : C \to I$ des villes aux installations ouvertes, tels que la somme des coûts de construction des installations ouvertes et des coûts de connexion des villes soit minimale.

Considérons le programme linéaire entier suivant pour ce problème. Dans ce programme, y_i est la variable indicatrice de l'ouverture de l'installation i, et x_{ij} est l'indicatrice de connexion de la ville j à l'installation i. Le premier jeu de contraintes garantit que chaque ville est reliée à au moins une installation, et le second assure que cette installation est bien ouverte.

[1] *Metric uncapacitated facility location*, en anglais.
[2] *Facilities*, en anglais.

$$\begin{aligned}
&\text{minimiser} && \sum_{i\in F,\ j\in C} c_{ij}x_{ij} + \sum_{i\in F} f_i y_i && && (24.1)\\
&\text{sous les contraintes} && \sum_{i\in F} x_{ij} \geqslant 1, && j \in C\\
& && y_i - x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && x_{ij} \in \{0,1\}, && i \in F,\ j \in C\\
& && y_i \in \{0,1\}, && i \in F
\end{aligned}$$

En voici la relaxation linéaire :

$$\begin{aligned}
&\text{minimiser} && \sum_{i\in F,\ j\in C} c_{ij}x_{ij} + \sum_{i\in F} f_i y_i && && (24.2)\\
&\text{sous les contraintes} && \sum_{i\in F} x_{ij} \geqslant 1, && j \in C\\
& && y_i - x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && y_i \geqslant 0, && i \in F
\end{aligned}$$

et son dual :

$$\begin{aligned}
&\text{maximiser} && \sum_{j\in C} \alpha_j && && (24.3)\\
&\text{sous les contraintes} && \alpha_j - \beta_{ij} \leqslant c_{ij}, && i \in F,\ j \in C\\
& && \sum_{j\in C} \beta_{ij} \leqslant f_i, && i \in F\\
& && \alpha_j \geqslant 0, && j \in C\\
& && \beta_{ij} \geqslant 0, && i \in F,\ j \in C
\end{aligned}$$

24.1 Une interprétation intuitive du dual

Familiarisons-nous tout d'abord avec la façon dont les variables duales « paient » pour une solution primale en nous plaçant dans le cadre suivant. Supposons que le programme (24.2) admette une solution optimale $I \subseteq F$ et $\phi : C \to I$, qui soit entière. Suivant cette hypothèse, $y_i = 1$ ssi $i \in I$, et $x_{ij} = 1$ ssi $i = \phi(j)$. Notons $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ une solution duale optimale.

Les conditions des écarts complémentaires donnent :

(i) $\forall i \in F, j \in C : x_{ij} > 0 \Rightarrow \alpha_j - \beta_{ij} = c_{ij}$

(ii) $\forall i \in F : y_i > 0 \Rightarrow \sum_{j\in C} \beta_{ij} = f_i$

(iii) $\forall j \in C : \alpha_j > 0 \Rightarrow \sum_{i \in F} x_{ij} = 1$

(iv) $\forall i \in F, j \in C : \beta_{ij} > 0 \Rightarrow y_i = x_{ij}$

La condition (ii) impose que le coût de construction de chaque installation ouverte soit payé intégralement, c'est-à-dire si $i \in I$, alors

$$\sum_{j\,:\,\phi(j)=i} \beta_{ij} = f_i.$$

Étudions la condition (iv). Si l'installation i est ouverte et que $\phi(j) \neq i$, alors $y_i \neq x_{ij}$ et donc $\beta_{ij} = 0$, c'est-à-dire la ville j ne contribue qu'au coût de construction de l'installation à laquelle elle est reliée.

D'après la condition (i), si $\phi(j) = i$ alors $\alpha_j - \beta_{ij} = c_{ij}$. Ainsi, α_j s'interprète comme la somme totale payée par la ville j : c_{ij} paie pour l'utilisation de l'arête ij ; et β_{ij} est la contribution de j à la construction de i.

24.2 Relaxation des conditions primales des écarts complémentaires

Une première idée serait de relâcher les conditions primales des écarts complémentaires de la façon suivante, sans modifier les conditions duales :

$$\forall j \in C : \quad c_{\phi(j)j}/3 \leqslant \alpha_j - \beta_{\phi(j)j} \leqslant c_{\phi(j)j},$$

et

$$\forall i \in I : \quad f_i/3 \leqslant \sum_{j:\ \phi(j)=i} \beta_{ij} \leqslant f_i.$$

Ainsi, nous obtiendrions une solution (entière) dont le coût est inférieur au triple de celui de la solution duale trouvée, d'où une 3-approximation. Cependant, nous souhaitons obtenir l'inégalité plus forte du théorème 24.7, où le dual paie au moins le tiers du coût de connexion, et paie totalement le coût d'ouverture des installations. Cette inégalité plus forte sera utile pour appliquer cet algorithme à la résolution du problème de la k-médiane au chapitre 25.

Pour cette raison, nous relâchons les conditions primales comme suit. Les villes sont réparties en deux catégories dans l'algorithme : les villes connectées *directement* ou *indirectement*. Seules les villes connectées directement paient pour l'ouverture des installations, c'est-à-dire $\beta_{ij} > 0$ uniquement si j est une ville directement connectée et si $i = \phi(j)$. Pour chaque ville j connectée indirectement, la condition primale est relâchée ainsi :

$$c_{\phi(j)j}/3 \leqslant \alpha_j \leqslant c_{\phi(j)j}.$$

Les autres conditions primales ne sont pas relâchées, c'est-à-dire pour chaque ville j connectée directement,

$$\alpha_j - \beta_{\phi(j)j} = c_{\phi(j)j},$$

et pour chaque installation ouverte i,

$$\sum_{j\,:\,\phi(j)=i} \beta_{ij} = f_i.$$

24.3 Algorithme primal-dual

L'algorithme procède en deux étapes. Durant la première étape, l'algorithme suit le schéma primal-dual. Il trouve une solution duale réalisable ainsi qu'un ensemble d'arêtes saturées et un ensemble F_t d'installations à ouvrir a priori. La deuxième étape détermine le sous-ensemble I d'installations de F_t à ouvrir définitivement et relie chaque ville à une installation de I.

Algorithme 24.2

Première étape
Nous recherchons une solution duale ayant la plus grande valeur possible. Nous utiliserons le processus suivant pour traiter les deux programmes linéaires (qui ne forment pas une paire couverture-empaquetage). Chaque ville j augmente sa variable duale α_j jusqu'à ce qu'elle soit connectée à une installation ouverte. L'ensemble des autres variables primales et duales ne font que suivre ce changement, pour essayer de maintenir la réalisabilité ou de satisfaire les conditions des écarts complémentaires.

Une notion de *temps* est introduite durant cette étape : chaque événement est daté au moment où il se produit. L'étape démarre au temps 0. Initialement, chaque ville est *déconnectée*. Tout au long de cette étape, l'algorithme augmente la variable duale α_j de chaque ville déconnectée j, uniformément, à vitesse unitaire, c'est-à-dire α_j croît de 1 par unité de temps. Dès que $\alpha_j = c_{ij}$ pour une arête (i, j), cette arête est déclarée *saturée*. À partir de cet instant, la variable duale β_{ij} est augmentée uniformément, garantissant ainsi que la première contrainte du programme (24.3) soit satisfaite. β_{ij} s'engage à payer pour l'installation i. Toute arête (i, j) telle que $\beta_{ij} > 0$ est dite *spéciale*.

Nous dirons que le coût de l'installation i est *couvert* si $\sum_j \beta_{ij} = f_i$. Dans ce cas, l'algorithme déclare que cette installation est *ouverte a priori*. De plus, toutes les villes déconnectées ayant une arête saturée vers cette installation sont déclarées *connectées* et l'installation i est le *témoin de connexion* de chacune de ces villes — remarquons que les variables duales α_j de ces villes ne seront plus augmentées. Par la suite, dès qu'une ville déconnectée j possède

une arête saturée vers i, j est déclarée connectée et i devient le témoin de connexion de j (remarquons qu'alors $\beta_{ij} = 0$, donc l'arête (i, j) n'est pas spéciale). La première étape s'arrête lorsque toutes les villes sont connectées. Si plusieurs événements se produisent simultanément, l'algorithme les traitent dans un ordre arbitraire.

Remarque 24.3 À la fin de la première étape, une ville peut avoir payé pour plusieurs installations ouvertes a priori. Cependant, nous devons garantir que chaque ville ne paie que pour l'installation à laquelle elle sera connectée finalement. Cela est assuré par la deuxième étape, qui détermine un sous-ensemble des installations ouvertes a priori pour les ouvrir définitivement.

Deuxième étape
Notons F_t l'ensemble des installations ouvertes a priori et T le sous-graphe des arêtes spéciales de G. Notons T^2 le graphe qui a l'arête (u, v) ssi il existe un chemin de longueur inférieure (ou égale) à 2 entre u et v dans T, et appelons H le sous-graphe de T^2 induit par F_t. Soit I un indépendant maximal (quelconque) de H. Toutes les installations de I sont déclarées *ouvertes*.

Pour chaque ville j, posons $\mathcal{F}_j = \{i \in F_t : (i, j) \text{ est spéciale}\}$. Comme I est indépendant, une au plus des installations de $\mathcal{F}_j$ est ouverte. Si une des installation $i \in \mathcal{F}_j$ est ouverte, alors posons $\phi(j) = i$ et la ville j est dite *connectée directement*. Sinon, considérons une arête saturée (i', j) où i' est le témoin de connexion de j. Si $i' \in I$, alors posons $\phi(j) = i'$ et la ville j est dite *connectée directement* (remarquons qu'alors $\beta_{i'j} = 0$). Sinon $i' \notin I$; soit i un voisin arbitraire de i' dans H tel que $i \in I$. Posons $\phi(j) = i$ et la ville j est dite *connectée indirectement*.

I et ϕ définissent une solution primale entière : $x_{ij} = 1$ ssi $\phi(j) = i$ et $y_i = 1$ ssi $i \in I$. Les valeurs de α_j et β_{ij} obtenues à la fin de la première étape forment une solution duale réalisable.

24.4 Analyse

Nous allons maintenant démontrer comment les variables duales (α_j) paient pour les coûts primaux d'ouverture des installations et de connexions des villes. Notons respectivement α_j^f et α_j^e les contributions de la ville j à ces deux coûts : $\alpha_j = \alpha_j^f + \alpha_j^e$. Si j est connectée indirectement, alors $\alpha_j^f = 0$ et $\alpha_j^e = \alpha_j$. Si j est connectée directement, alors nous savons que :

$$\alpha_j = c_{ij} + \beta_{ij},$$

où $i = \phi(j)$. Posons donc $\alpha_j^f = \beta_{ij}$ et $\alpha_j^e = c_{ij}$.

Lemme 24.4 *Pour tout $i \in I$,*

$$\sum_{j\,:\,\phi(j)=i} \alpha_j^f = f_i.$$

Preuve : Comme i est ouverte a priori à la fin de la première étape, ses frais d'ouverture sont couverts, c'est-à-dire

$$\sum_{j\,:\,(i,j)\text{ est spéciale}} \beta_{ij} = f_i.$$

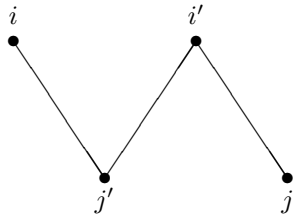
L'observation clé est que toutes les villes j qui ont contribué à f_i sont directement connectées à i. Pour chacune de ces villes, $\alpha_j^f = \beta_{ij}$. Enfin, pour toute autre ville j' connectée à l'installation i, nous avons $\alpha_{j'}^f = 0$. CQFD. □

Corollaire 24.5 $\sum_{i\in I} f_i = \sum_{j\in C} \alpha_j^f$.

Souvenons-nous que $\alpha_j^f = 0$ par définition pour les villes connectées indirectement. Ainsi, les frais d'ouverture des installations sont payés exclusivement par les villes connectées directement.

Lemme 24.6 *Pour toute ville j connectée indirectement, $c_{ij} \leqslant 3\alpha_j^e$, où $i = \phi(j)$.*

Preuve : Soit i' le témoin de connexion de la ville j. Comme j est connectée indirectement à i, (i, i') est une arête de H. Ainsi, il existe une ville j' telle que (i, j') et (i', j') soient des arêtes spéciales. Soient t_1 et t_2 les dates auxquelles i et i' ont été ouvertes a priori durant la première étape.



Comme l'arête (i', j) est saturée, $\alpha_j \geqslant c_{i'j}$. Démontrons que $\alpha_j \geqslant c_{ij'}$ et $\alpha_j \geqslant c_{i'j'}$. Le lemme s'ensuivra par l'inégalité triangulaire.

Comme les arêtes (i', j') et (i, j') sont saturées, $\alpha_{j'} \geqslant c_{ij'}$ et $\alpha_{j'} \geqslant c_{i'j'}$. Puisque ces arêtes sont toutes deux spéciales, elles ont saturé avant que i ou i' ne soient ouvertes a priori. Plaçons-nous au temps $\min(t_1, t_2)$. Par construction, $\alpha_{j'}$ n'augmente plus après cette date. Ainsi, $\alpha_{j'} \leqslant \min(t_1, t_2)$. Enfin, puisque i' est le témoin de connexion de j, $\alpha_j \geqslant t_2$. Nous en concluons que $\alpha_j \geqslant \alpha_{j'}$, et l'inégalité souhaitée s'ensuit. □

Théorème 24.7 *Les solutions primale et duale construites par l'algorithme satisfont :*

$$\sum_{i\in F,\, j\in C} c_{ij}x_{ij} + 3\sum_{i\in F} f_i y_i \leqslant 3\sum_{j\in C} \alpha_j.$$

Preuve : Pour chaque ville directement connectée j, $c_{ij} = \alpha_j^e \leqslant 3\alpha_j^e$, où $\phi(j) = i$. Le lemme 24.6 donne donc :

$$\sum_{i\in F,\, j\in C} c_{ij}x_{ij} \leqslant 3\sum_{j\in C} \alpha_j^e.$$

On obtient le théorème en ajoutant l'égalité établie au corollaire 24.5 multipliée par 3. □

24.4.1 Évaluation du temps de calcul

Un avantage du schéma primal-dual est qu'il produit des algorithmes rapides. C'est typiquement le cas pour l'implémentation suivante de l'algorithme. Posons : $n_c = |C|$ et $n_f = |F|$. Le nombre total de sommets est $n_c + n_f = n$, et le nombre total d'arêtes $n_c \times n_f = m$.

Trions l'ensemble des arêtes par coût croissant — nous savons ainsi dans quel l'ordre et quand chaque arête saturera. À chaque installation i, nous associons le nombre courant de villes qui contribuent à son coût, et la *date prévue* t_i où ses frais seront couverts si aucun autre événement ne se produit entre temps. Initialement, tous les (t_i) sont infinis, et chaque installation n'a aucune ville participant à son coût. Les t_i sont rangés dans un tas binaire, afin de pouvoir les mettre à jour et trouver le minimum courant en temps $O(\log n_f)$. Deux types d'événements sont possibles et conduisent aux mises à jour suivantes.

- Une arête (i, j) sature.
  - Si l'installation i n'est pas ouverte a priori, alors elle reçoit une ville de plus finançant son coût. La contribution courante totale à son coût se calcule aisément, et la date prévue pour l'installation i est recalculée en temps constant. Le tas est alors mis à jour en temps $O(\log n_f)$.
  - Si l'installation i est déjà ouverte a priori, la ville j est déclarée connectée, et α_j n'augmente plus. Le nombre de contribuables pour chaque installation i' qui comptait j parmi ses contribuables, doit donc être diminué de 1, et sa date prévue de recouvrement recalculée.
- Les frais de l'installation i sont couverts. Dans ce cas, i est déclarée ouverte a priori, et toutes les villes qui contribuent pour i sont déclarées connectées. Pour chacune de ces villes, on exécute le deuxième point

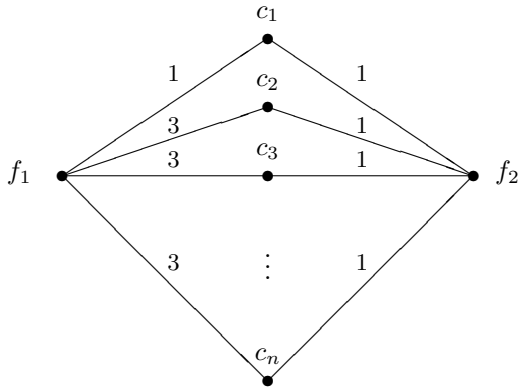
de l'événement précédent, c'est-à-dire on met à jour les installations auxquelles elles contribuaient.

Le théorème suivant s'obtient en constatant que chaque arête ne sera examinée qu'au plus deux fois : premièrement, quand elle sera saturée ; deuxièmement, quand la ville j sera connectée. Et chacun de ces examens prend un temps $O(\log n_f)$.

Théorème 24.8 *L'algorithme 24.2 est une* 3*-approximation pour le problème du placement d'installations et s'exécute en temps* $O(m \log m)$.

24.4.2 Instance critique

La famille infinie d'instances suivante démontre que l'analyse de l'algorithme est exacte. Considérons un graphe à n villes $c_1, c_2, \ldots, c_n$ et deux installations f_1 et f_2. Chaque ville est à distance 1 de f_2. La ville c_1 est à distance 1 de f_1, et $c_2, \ldots, c_n$ sont à distance 3 de f_1. Les coûts d'ouverture de f_1 et f_2 sont respectivement ε et $(n+1)\varepsilon$, pour un ε suffisamment petit.



La solution optimale ouvre f_2 et y connecte toutes les villes, pour un coût total $(n+1)\varepsilon + n$. Cependant, l'algorithme 24.2 ouvre f_1 et y connecte toutes les villes, pour un coût total $\varepsilon + 1 + 3(n-1)$.

24.5 Exercices

24.1 Considérons le problème général du placement d'installations, où les coûts de connexion ne sont pas métriques. Proposez une réduction à partir du problème de la couverture par ensembles pour démontrer qu'approcher ce problème à un facteur garanti est aussi difficile qu'approcher la couverture par ensembles et donc qu'on ne peut pas espérer un facteur d'approximation inférieur à $\Theta(\log n)$, à moins que $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log \log n)})$

(voir chapitre 29). Puis, proposez une $O(\log n)$-approximation pour ce problème.

24.2 Modifiez la deuxième étape de l'algorithme pour sélectionner dans T les arêtes saturées au lieu des arêtes spéciales. Démontrez que le lemme 24.6 devient faux. Modifiez l'algorithme encore une fois pour obtenir un lemme équivalent au lemme 24.6.
Indication : Triez les installation de H dans l'ordre où elles ont été ouvertes a priori, et sélectionnez dans I l'ensemble indépendant maximal le plus petit pour l'ordre lexicographique.

24.3 Exhibez une instance critique à un facteur 3 pour l'algorithme 24.2, où les ensembles des villes et des installations sont identiques, c'est-à-dire où $C = F$.

24.4 Considérez la preuve du lemme 24.6 et exhibez une instance telle que $\alpha_j > t_2$.

24.5 Le vecteur $\boldsymbol{\alpha}$ construit par l'algorithme 24.2 est maximal dans le sens suivant : si on augmente n'importe quel α_j de ce vecteur, alors il n'existe aucune valeur des (β_{ij}) qui donne une solution réalisable. Toutes les solutions maximales $\boldsymbol{\alpha}$ sont-elles à un facteur 3 du coût dual optimal ?
Indication : Il est facile de construire une solution maximale qui coûte $2/n$ fois le coût optimal seulement. Prenons n installations de coût d'ouverture unitaire et n villes connectées chacune à une installation distincte par des arêtes de coût ε. Une ville supplémentaire est connectée à chacune des installations par une arête de coût unitaire.

24.6 Étudions la variante suivante du problème du placement d'installations métrique sans capacité. Ici, la connexion de la ville j à l'installation i coûte c_{ij}^2. Les (c_{ij}) vérifient l'inégalité triangulaire (mais pas les nouveaux coûts de connexion (c_{ij}^2)). Démontrez que l'algorithme 24.2 est une 9-approximation pour ce problème.

24.7 Considérons la généralisation suivante à des demandes arbitraires. Chaque ville j est munie d'une demande d_j, et n'importe quelle installation peut satisfaire cette demande. La satisfaction de cette demande par l'installation i coûte $c_{ij}d_j$. Proposez un programme linéaire entier et sa relaxation pour ce problème, et modifiez l'algorithme 24.2 pour obtenir une 3-approximation.
Indication : Augmenter chaque α_j à un taux d_j.

24.8 Dans le problème du placement d'installations *avec capacités*,[3] chaque installation i a une capacité u_i, et peut donc servir jusqu'à u_i villes, mais

[3] *Metric capacitated facility location*, en anglais.

pas plus. Démontrez que le saut intégral de l'adaptation naturelle du programme (24.2) à ce problème n'est pas borné.

24.9 Étudions la variante du problème du placement d'installations métrique avec capacités, où chaque installation peut être ouverte un nombre de fois arbitraire. Si l'installation i est ouverte y_i fois, alors elle peut servir jusqu'à $u_i y_i$ villes. Donnez un programme linéaire entier et sa relaxation pour ce problème, puis modifiez l'algorithme 24.2 pour obtenir une approximation à un facteur constant.

24.10 (Charikar, Khuller, Mount et Narshimhan [42]) Étudions la variante *avec pénalités*[4] du problème du placement d'installations, où l'on doit payer une pénalité pour chaque ville qui n'est pas connectée à une installation ouverte. Le but est de minimiser la somme des coûts de connexion, d'ouverture des installations et des pénalités. Proposez une 3-approximation pour ce problème.

24.11 (Jain et Vazirani [148]) Considérons la variante du problème du placement d'installations *avec tolérance aux pannes*,[5] où chaque ville j doit être connectée à r_j installations ouvertes distinctes. Le but est, bien entendu, de minimiser la somme des coûts de connexion et d'ouverture des installations.

Décomposez le problème en k étapes, numérotées à rebours de k à 1, à l'instar de l'exercice 23.7. Durant l'étape p, toutes les villes qui ont une requête de connexion résiduelle p sont connectées à une ou plusieurs installations ouvertes. L'étape p consiste à exécuter l'algorithme présenté dans ce chapitre sur le graphe G_p suivant. Le coût de chaque installation ouverte lors d'une étape précédente est mis à 0. Si la ville j est connectée à l'installation i lors d'une étape précédente, alors c_{ij} est mis à ∞.

1. Démontrez que même si G_p ne respecte pas l'inégalité triangulaire en quelques endroits, cet algorithme produit une solution de coût inférieur au triple de l'optimum pour ce graphe.

   Indication : Chaque fois qu'il est nécessaire de court-circuiter une arête, l'inégalité triangulaire est vérifiée.

2. Démontrez que le coût de la solution calculée à l'étape p est inférieur à $3 \cdot \mathrm{OPT}/p$, où OPT est le coût optimal d'une solution au problème total.

   Indication : Ôtez d'une solution optimale les arêtes de coût ∞ dans G_p, et divisez le reste par p. Démontrez que c'est une solution fractionnaire réalisable pour l'étape p.

3. Concluez que cet algorithme est une $3 \cdot H_k$-approximation pour le problème du placement d'installations avec tolérance aux pannes.

[4] *Metric prize-collecting facility location problem*, en anglais.
[5] *Metric fault tolerant facility location problem*, en anglais.

24.12 (Mahdian, Markakis, Saberi, et Vazirani [209]) Cet exercice propose une 3-approximation gloutonne pour le problème du placement d'installations métrique sans capacité, ainsi que son analyse par la méthode de l'alignement dual.

Modifions l'algorithme 24.2 comme suit. Comme précédemment, les variables duales (α_j) de toutes les villes non connectées j sont augmentées uniformément. Dès qu'une arête (i, j) est saturée, β_{ij} est augmentée. Dès que les frais d'ouverture d'une installation i sont couverts, cette installation est ouverte. Notons S l'ensemble des villes non connectées ayant des arêtes saturées vers i. Chaque ville $j \in S$ est alors déclarée connectée et n'augmente plus son α_j. Jusqu'à présent, cet algorithme est identique à l'algorithme 24.2. La différence principale intervient ici : chaque ville $j \in S$ retire sa contribution aux autres installations, c'est-à-dire pour chaque installation $i' \neq i$, on pose $\beta_{i'j} = 0$. Quand toutes les villes sont déclarées connectées, l'algorithme s'arrête. Remarquez que chaque ville contribue au coût d'ouverture d'une unique installation — celle à laquelle elle sera connectée.

1. Cet algorithme peut se décrire très simplement de façon gloutonne. Donnez cette description.

   Indication : Utilisez la notion de coût effectif définie dans l'algorithme glouton pour la couverture par ensembles.

2. Les trois parties suivantes utilisent la méthode de l'alignement dual pour analyser cet algorithme. Commencez par observer que la solution primale obtenue est intégralement payée par la solution duale calculée.

3. Soit i une installation ouverte et $\{1, \ldots, k\}$ l'ensemble des villes qui ont contribué à l'ouverture de i à un moment ou un autre de l'algorithme. Supposons sans perte de généralité que $\alpha_1 \leqslant \alpha_j$, pour $j \leqslant k$. Montrez que pour $j \leqslant k$, $\alpha_j - c_{ij} \leqslant 2\alpha_1$. Puis, montrez que :

   $$\sum_{j=1}^{k} \alpha_j \leqslant 3 \sum_{j=1}^{k} c_{ij} + f_i.$$

   Indication : En sus de l'inégalité triangulaire, utilisez l'inégalité suivante, qui est la conséquence qu'à tout instant, la contribution totale à l'ouverture de chaque installation i est inférieure à f_i :

   $$\sum_{j \,:\, c_{ij} \leqslant \alpha_1} (\alpha_1 - c_{ij}) \leqslant f_i.$$

4. Déduisez-en que $\boldsymbol{\alpha}/3$ est une solution duale réalisable.

5. Comment l'analyse pourrait-elle être améliorée ? — on sait démontrer que cet algorithme est une 1.86-approximation.

6. Proposez une implémentation efficace de cet algorithme, ayant le même temps d'exécution que l'algorithme 24.2

7. Voyez-vous où améliorer encore cet algorithme ?
   Indication : Imaginez que la ville j soit connectée à une installation ouverte i à un moment donné dans l'algorithme. Par la suite, une installation i' est ouverte, et supposez que $c_{ij} > c_{i'j}$. Alors, connecter j à i' réduira le coût de la solution.

24.13 (Mahdian, Markakis, Saberi et Vazirani [209]) Prenons la variante suivante du problème du placement d'installations métrique sans capacité. Au lieu de f_i, le coût d'ouverture de chaque installation $i \in F$ est affine, avec un coût initial s_i et un coût incrémental t_i. Le coût d'ouverture de l'installation i pour y connecter $k > 0$ villes sera $s_i + kt_i$. Les coûts de connexion sont comme précédemment métriques. Le but est toujours de connecter chaque ville à une installation ouverte, de façon à minimiser la somme des coûts de connexion et d'ouverture. Proposez une réduction isofacteur de ce problème au problème du placement d'installation métrique sans capacité.
Indication : Modifiez la métrique convenablement.

24.6 Notes

Le premier algorithme d'approximation pour le placement d'installation métrique sans capacité est dû à Hochbaum [132] et réalisait une $O(\log n)$-approximation. Le premier algorithme d'approximation à un facteur constant, une 3.16-approximation, a été établi par Shmoys, Tardos et Aardal [248]. Il fonctionnait par arrondi de la solution d'un programme linéaire. Le résultat de ce chapitre est dû à Jain et Vazirani [149]. La meilleure approximation connue aujourd'hui est une 1.52-approximation, due à Mahdian, Ye et Zhang [210]. D'autre part, Guha et Khuller [125] ont démontré à partir du résultat d'inapproximabilité de la couverture par ensembles (voir chapitre 29) qu'il était impossible d'obtenir un facteur d'approximation inférieur à 1.463, à moins que $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log \log n)})$.

25 *k*-Médiane

Le problème de la k-médiane[1] diffère du problème du placement d'installations sur deux points essentiels : l'ouverture d'une installation ne coûte rien, et on s'impose d'ouvrir au plus k installations. Ce problème modélise la recherche d'une classification[2] de coût minimum (en moins de k classes), et a donc de très nombreuses applications.

Le principe du schéma primal-dual est d'améliorer localement et judicieusement la solution. Il ne n'est donc pas idéal lorsque les contraintes sont globales, comme dans le problème de la k-médiane où seules k installations peuvent être ouvertes. Nous contournons cette difficulté en utilisant une technique très puissante en optimisation combinatoire, la relaxation lagrangienne.

Problème 25.1 (k-Médiane métrique) Soient $G = ((F, C), E)$ un graphe biparti, où F est l'ensemble des *installations*, C l'ensemble des *villes*, et $k > 0$ un entier spécifiant le nombre maximum d'installations qu'on s'autorise à ouvrir. On note c_{ij} le coût de connexion de la ville j à l'installation i. Les coûts de connexion vérifient l'inégalité triangulaire. Le problème est de trouver un sous-ensemble $I \subseteq F$ d'installations à ouvrir, avec $|I| \leqslant k$, et une fonction $\phi : C \to I$ associant chaque ville à une installation ouverte, de façon à minimiser le coût total des connexions.

25.1 Relaxation et dual

Le problème de la k-médiane est équivalent au programme linéaire entier suivant. Les variables indicatrices (y_i) et (x_{ij}) y jouent les mêmes rôles que dans le programme (24.1).

[1] *k-Median problem*, en anglais.
[2] *Clustering*, en anglais.

$$\begin{aligned}
&\text{minimiser} && \sum_{i\in F,\ j\in C} c_{ij}x_{ij} && && (25.1)\\
&\text{sous les contraintes} && \sum_{i\in F} x_{ij} \geqslant 1, && j \in C\\
& && y_i - x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && \sum_{i\in F} -y_i \geqslant -k\\
& && x_{ij} \in \{0,1\}, && i \in F,\ j \in C\\
& && y_i \in \{0,1\}, && i \in F
\end{aligned}$$

Sa relaxation est :

$$\begin{aligned}
&\text{minimiser} && \sum_{i\in F,\ j\in C} c_{ij}x_{ij} && && (25.2)\\
&\text{sous les contraintes} && \sum_{i\in F} x_{ij} \geqslant 1, && j \in C\\
& && y_i - x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && \sum_{i\in F} -y_i \geqslant -k\\
& && x_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && y_i \geqslant 0, && i \in F
\end{aligned}$$

et son dual :

$$\begin{aligned}
&\text{maximiser} && \sum_{j\in C} \alpha_j - zk && && (25.3)\\
&\text{sous les contraintes} && \alpha_j - \beta_{ij} \leqslant c_{ij}, && i \in F,\ j \in C\\
& && \sum_{j\in C} \beta_{ij} \leqslant z, && i \in F\\
& && \alpha_j \geqslant 0, && j \in C\\
& && \beta_{ij} \geqslant 0, && i \in F,\ j \in C\\
& && z \geqslant 0
\end{aligned}$$

25.2 Principe de l'algorithme

Du fait des ressemblances entre les deux problèmes, le placement d'installations et la k-médiane, leurs programmes linéaires présentent des similitudes que nous allons exploiter. Prenons une instance du problème de la k-médiane, donnons un coût z à l'ouverture de chacune des installations, puis recherchons

des solutions optimales, $(\boldsymbol{x}, \boldsymbol{y})$ et $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, aux programmes (24.2) et (24.3), respectivement. D'après le théorème de dualité forte,

$$\sum_{i \in F,\, j \in C} c_{ij} x_{ij} + \sum_{i \in F} z y_i \;=\; \sum_{j \in C} \alpha_j .$$

Supposons à présent que la solution primale $(\boldsymbol{x}, \boldsymbol{y})$ ouvre exactement k installations (éventuellement de façon fractionnaire), c'est-à-dire que $\sum_i y_i = k$. Alors, nous affirmons que $(\boldsymbol{x}, \boldsymbol{y})$ et $(\boldsymbol{\alpha}, \boldsymbol{\beta}, z)$ sont des solutions optimales des programmes (25.2) et (25.3), respectivement. Elles sont clairement réalisables. Et pour l'optimalité, il suffit de substituer $\sum_i y_i = k$ dans l'égalité ci-dessus et de réorganiser les termes pour montrer que les solutions primale et duale ont la même valeur objectif :

$$\sum_{i \in F,\, j \in C} c_{ij} x_{ij} \;=\; \sum_{j \in C} \alpha_j - zk.$$

Exploitons donc cette idée, couplée avec l'algorithme 24.2 et le théorème 24.7, pour obtenir une « bonne » solution entière du programme (25.2). Supposons qu'avec un coût z pour l'ouverture de chaque installation, l'algorithme 24.2 produise des solutions $(\boldsymbol{x}, \boldsymbol{y})$ et $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, où la solution primale ouvre exactement k installations. D'après le théorème 24.7,

$$\sum_{i \in F,\, j \in C} c_{ij} x_{ij} + 3zk \;\leqslant\; 3 \sum_{j \in C} \alpha_j .$$

Or, remarquons que $(\boldsymbol{x}, \boldsymbol{y})$ et $(\boldsymbol{\alpha}, \boldsymbol{\beta}, z)$ sont des solutions réalisables, respectivement primale (entière) et duale du problème de la k-médiane, qui satisfont :

$$\sum_{i \in F,\, j \in C} c_{ij} x_{ij} \;\leqslant\; 3\Big(\sum_{j \in C} \alpha_j - zk\Big).$$

Par conséquent, $(\boldsymbol{x}, \boldsymbol{y})$ est une solution du problème de la k-médiane coûtant moins du triple de l'optimum.

Remarquons que cette analyse à un facteur 3 de l'optimum n'est pas correcte si moins de k installations sont ouvertes ; si plus de k installations étaient ouvertes, la solution ne serait pas réalisable pour la k-médiane. Il reste donc à déterminer une bonne valeur de z qui ouvrira *exactement* k installations. Plusieurs idées sont nécessaires. La première s'inspire de l'économie : taxer permet de contrôler la quantité de biens traversant une frontière – augmenter les taxes réduit le flot et vice-versa. Ainsi, augmenter z devrait réduire le nombre d'installations ouvertes et vice-versa.

Il est donc naturel de chercher à modifier l'algorithme 24.2 pour qu'il trouve une valeur de z telle qu'exactement k installations soient ouvertes. Cela conduirait à une 3-approximation. Cependant, on ne connaît pas de telle

modification. Nous allons présenter à la place, une stratégie qui donne une 6-approximation. À partir de maintenant, nous supposerons que nous n'avons pas obtenu d'exécution de l'algorithme qui ouvre exactement k installations.

Clairement, si $z = 0$, l'algorithme ouvre toutes les installations, et si z est très grand, il n'en ouvrira qu'une. Pour le deuxième cas, il suffit de poser $z = nc_{\max}$, où $c_{\max}$ est la longueur maximale d'une arête. Nous procédons donc par recherche dichotomique dans l'intervalle $[0, nc_{\max}]$ pour trouver z_1 et z_2 tels que l'algorithme ouvre $k_2 > k$ et $k_1 < k$ installations, respectivement, et tels que $z_1 - z_2 \leqslant c_{\min}/(12n_c^2)$, où $c_{\min}$ est la longueur de la plus petite arête non nulle. Comme précédemment, notons $n_c = |C|$ et $n_f = |F|$. Le nombre total de sommets est $n_c + n_f = n$ et le nombre total d'arêtes est $n_c \times n_f = m$. Soient $(\boldsymbol{x}^s, \boldsymbol{y}^s)$ et $(\boldsymbol{x}^l, \boldsymbol{y}^l)$ les deux solutions primales trouvées, avec $\sum_{i\in F} y_i^s = k_1$ et $\sum_{i\in F} y_i^l = k_2$ (les lettres en exposant signifient s pour *small* et l pour *large*[3]). Enfin, notons $(\boldsymbol{\alpha}^s, \boldsymbol{\beta}^s)$ et $(\boldsymbol{\alpha}^l, \boldsymbol{\beta}^l)$, les solutions duales correspondantes trouvées.

Soit $(\boldsymbol{x}, \boldsymbol{y}) = a(\boldsymbol{x}^s, \boldsymbol{y}^s) + b(\boldsymbol{x}^l, \boldsymbol{y}^l)$ la combinaison convexe des deux solutions, telle que $ak_1 + bk_2 = k$. En fait, $a = (k_2 - k)/(k_2 - k_1)$ et $b = (k-k_1)/(k_2-k_1)$. Puisque $(\boldsymbol{x}, \boldsymbol{y})$ est une solution (fractionnaire) réalisable du problème du placement d'installations, qui ouvre exactement k installations, c'est également, une solution (fractionnaire) réalisable du problème de la k-médiane. Dans cette solution, chaque ville est connectée à une ou à deux installations.

Lemme 25.2 *Le coût de $(\boldsymbol{x}, \boldsymbol{y})$ est inférieur à $(3 + 1/n_c)$ fois le coût d'une solution fractionnaire optimale au problème de la k-médiane.*

Preuve : D'après le théorème 24.7, nous savons que

$$\sum_{i\in F,\, j\in C} c_{ij} x_{ij}^s \;\leqslant\; 3\Big(\sum_{j\in C} \alpha_j^s - z_1 k_1\Big),$$

et que

$$\sum_{i\in F,\, j\in C} c_{ij} x_{ij}^l \;\leqslant\; 3\Big(\sum_{j\in C} \alpha_j^l - z_2 k_2\Big).$$

Comme $z_1 > z_2$, $(\boldsymbol{\alpha}^l, \boldsymbol{\beta}^l)$ est une solution réalisable du dual du problème du placement d'installations, même si le coût des installations est z_1. Nous souhaitons remplacer z_2 par z_1 dans la seconde inégalité, quite à augmenter le facteur d'approximation. Nous y arrivons en utilisant la borne sur $z_1 - z_2$ et le fait que $\sum_{i\in F,\, j\in C} c_{ij} x_{ij}^l \;\geqslant\; c_{\min}$. Nous obtenons :

$$\left(3 + \frac{1}{n_c}\right)\left(\sum_{j\in C} \alpha_j^l - z_1 k_2\right).$$

[3] *petit* et *grand*, respectivement, en français.

Ajouter cette inégalité multipliée par b à la première inégalité multipliée par a donne :

$$\sum_{i\in F,\ j\in C} c_{ij}x_{ij} \ \leqslant\ \left(3+\frac{1}{n_c}\right)\left(\sum_{j\in C}\alpha_j - z_1 k\right),$$

où $\boldsymbol{\alpha} = a\boldsymbol{\alpha}^s + b\boldsymbol{\alpha}^l$. Posons $\boldsymbol{\beta} = a\boldsymbol{\beta}^s + b\boldsymbol{\beta}^l$. Remarquons que $(\boldsymbol{\alpha}, \boldsymbol{\beta}, z_1)$ est une solution réalisable du dual du problème de la k-médiane. CQFD. □

La section 25.3 propose une procédure randomisée d'arrondi pour obtenir une solution entière du problème de la k-médiane à partir de $(\boldsymbol{x}, \boldsymbol{y})$, en échange d'une légère augmentation du coût. Cette procédure sera dérandomisée section 25.3.1.

25.3 Arrondi randomisé

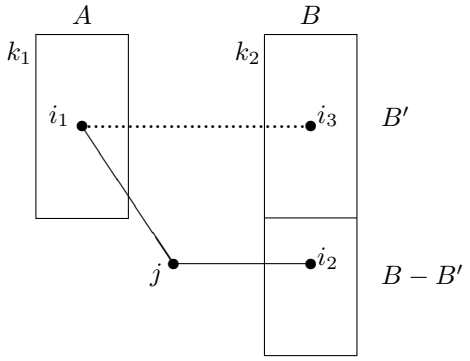
Nous présentons ici une procédure randomisée d'arrondi qui produit une solution entière du problème de la k-médiane à partir de $(\boldsymbol{x}, \boldsymbol{y})$. Cela induira une augmentation du coût d'un facteur $1 + \max(a, b)$.

Notons A et B les ensembles d'installations ouvertes respectivement par les deux solutions $|A| = k_1$ et $|B| = k_2$. Recherchons pour chaque installation de A, l'installation qui en est la plus proche dans B — ces installations ne sont pas nécessairement distinctes. Notons $B' \subset B$ l'ensemble de ces installations. Si $|B'| < k_1$, ajoutons des installations arbitraires supplémentaires de $B - B'$ à B' pour que $|B'| = k_1$.

Avec probabilité a, ouvrons toutes les installations de A, et avec probabilité $b = 1 - a$, ouvrons toutes les installations de B'. Ouvrons en sus un ensemble de $k - k_1$ installations choisies uniformément dans $B - B'$. Remarquons que chaque installation de $B - B'$ est ouverte avec probabilité b. Notons I l'ensemble des installations ouvertes, $|I| = k$.

La fonction $\phi : C \to I$ est définie ainsi. Considérons une ville j reliée à $i_1 \in A$ et $i_2 \in B$ dans les deux solutions. Si $i_2 \in B'$, alors i_1 ou i_2 sont ouvertes ci-dessus, i_1 avec probabilité a et i_2 avec probabilité b. Connectez la ville j à celle qui est ouverte.

Si $i_2 \in B - B'$, soit $i_3 \in B'$ l'installation de B la plus proche de i_1. Connectons j à i_2 si elle est ouverte. Sinon, connectons-la à i_1 si elle est ouverte. Enfin, si ni i_2 ni i_1 ne sont ouvertes, connectons-la à i_3.



Notons coût(j) le coût de connexion de la ville j dans la solution fractionnaire $(\boldsymbol{x}, \boldsymbol{y})$: coût(j) $= ac_{i_1 j} + bc_{i_2 j}$.

Lemme 25.3 *L'espérance* $\mathbf{E}[c_{\phi(j)j}]$ *du coût de connexion de la ville* j *dans la solution entière est inférieure à* $(1 + \max(a, b))$ coût(j). *De plus,* $\mathbf{E}[c_{\phi(j)j}]$ *se calcule efficacement.*

Preuve : Si $i_2 \in B'$, $\mathbf{E}[c_{\phi(j)j}] = ac_{i_1 j} + bc_{i_2 j} =$ coût(j). Étudions le deuxième cas, où $i_2 \notin B'$. Ici, i_2 est ouverte avec probabilité b. La probabilité que i_2 ne soit pas ouverte et que i_1 soit ouverte, est $(1 - b)a = a^2$, et la probabilité que ni i_2 ni i_1 soient ouvertes, est $(1 - b)(1 - a) = ab$. Ainsi,

$$\mathbf{E}[c_{\phi(j)j}] \leqslant bc_{i_2 j} + a^2 c_{i_1 j} + abc_{i_3 j}.$$

Puisque i_3 est l'installation de B la plus proche de i_1, $c_{i_1 i_3} \leqslant c_{i_1 i_2} \leqslant c_{i_1 j} + c_{i_2 j}$ (la seconde inégalité est due à l'inégalité triangulaire). Puis, par l'inégalité triangulaire encore, $c_{i_3 j} \leqslant c_{i_1 j} + c_{i_1 i_3} \leqslant 2c_{i_1 j} + c_{i_2 j}$. Ainsi,

$$\mathbf{E}[c_{\phi(j)j}] \leqslant bc_{i_2 j} + a^2 c_{i_1 j} + ab(2c_{i_1 j} + c_{i_2 j}).$$

Or, $a^2 c_{i_1 j} + abc_{i_1 j} = ac_{i_1 j}$. Donc,

$$\begin{aligned}\mathbf{E}[c_{\phi(j)j}] &\leqslant (ac_{i_1 j} + bc_{i_2 j}) + ab(c_{i_1 j} + c_{i_2 j}) \\ &\leqslant (ac_{i_1 j} + bc_{i_2 j})(1 + \max(a, b)).\end{aligned}$$

Clairement, $\mathbf{E}[c_{\phi(j)j}]$ se calcule rapidement dans les deux cas. □

Notons $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ la solution entière obtenue pour le problème de la k-médiane par cette procédure d'arrondi randomisé. Alors,

Lemme 25.4 $$\mathbf{E}\left[\sum_{i \in F,\, j \in C} c_{ij} x_{ij}^k\right] \leqslant (1 + \max(a, b)) \left(\sum_{i \in F,\, j \in C} c_{ij} x_{ij}\right)$$

et l'espérance du coût de la solution trouvée se calcule efficacement.

25.3.1 Dérandomisation

Il s'agit d'appliquer directement la méthode de l'espérance conditionnelle. Tout d'abord, l'algorithme randomisé ouvre les installations de A avec probabilité a et celles de B' avec probabilité $b = 1 - a$. Sélectionnons$A$, et calculons l'espérance de la valeur objectif si $k - k_1$ installations sont choisies uniformément dans $B - B'$. Ensuite, procédons de même en sélectionnant B' au lieu de A. Choisissons d'ouvrir l'ensemble dont l'espérance du coût est la plus faible.

L'algorithme ouvre ensuite un sous-ensemble aléatoire de $k - k_1$ installations de $B - B'$. Étant donné la sélection courante $D \subset B - B'$, $|D| \leqslant k - k_1$, notons $\mathbf{E}[D, B - (B' \cup D)]$ l'espérance du coût de la solution si toutes les installations de D sont ouvertes et si $k - k_1 - |D|$ autres installations sont choisies uniformément dans $B - (B' \cup D)$. Comme chaque installation de $B - (B' \cup D)$ a la même probabilité d'être ouverte, nous obtenons :

$$\mathbf{E}[D, B - (B' \cup D)] = \frac{1}{|B - (B' \cup D)|} \sum_{i \in B - (B' \cup D)} \mathbf{E}[D \cup \{i\}, B - (B' \cup D \cup \{i\})].$$

Ainsi, il existe un i tel que

$$\mathbf{E}[D \cup \{i\}, B - (B' \cup D \cup \{i\})] \leqslant \mathbf{E}[D, B - (B' \cup D)].$$

Choisissez un tel i et itérez en remplaçant D par $D \cup \{i\}$. Remarquez que le calcul de $\mathbf{E}[D \cup \{i\}, B - (B' \cup D \cup \{i\})]$ peut s'effectuer comme au lemme 25.4.

25.3.2 Temps d'exécution

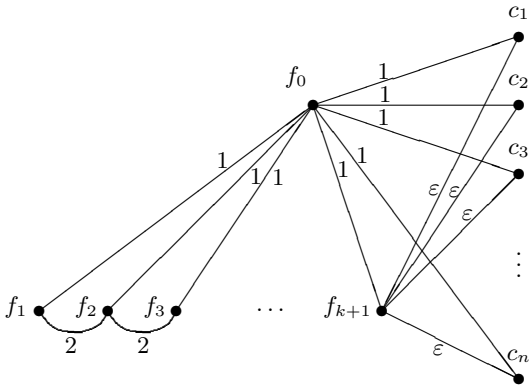
On remarque facilement que $a \leqslant 1 - 1/n_c$ (c'est le cas pour $k_1 = k - 1$ et $k_2 = n_c$) et que $b \leqslant 1 - 1/k$ (c'est le cas pour $k_1 = 1$ et $k_2 = k + 1$). Ainsi, $1 + \max(a, b) \leqslant 2 - 1/n_c$. En reprenant tout, nous obtenons le facteur d'approximation suivant : $(2 - 1/n_c)(3 + 1/n_c) < 6$. Cette procédure se dérandomise par la méthode de l'espérance conditionnelle (voir section 25.3.1). La recherche dichotomique effectue $O(\log_2(n^3 c_{\max}/c_{\min})) = O(L + \log n)$ tests, où $L = \log(c_{\max}/c_{\min})$. Le temps d'exécution de chaque test est dominé par le temps de calcul de l'algorithme 24.2 ; l'arrondi randomisé prend un temps $O(n)$, et sa dérandomisation un temps $O(m)$. Nous concluons donc le théorème suivant.

Théorème 25.5 *L'algorithme décrit ci-dessus est une* 6*-approximation pour le problème de la k-médiane. Son temps d'exécution est* $O(m \log m\,(L + \log n))$.

25.3.3 Instance critique

On ne connaît pas d'instance critique à un facteur 6 pour cet algorithme pour la k-médiane. Cependant, la famille d'instances suivante démontre que l'analyse de la procédure d'arrondi ne peut pas être améliorée.

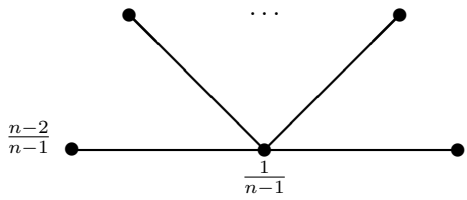
Les deux solutions $(\boldsymbol{x}^s, \boldsymbol{y}^s)$ et $(\boldsymbol{x}^l, \boldsymbol{y}^l)$ ouvrent, respectivement, une installation f_0 et $k+1$ installations $f_1, \ldots, f_{k+1}$. La distance entre f_0 et chaque f_i, $i \geqslant 1$, vaut 1. La distance entre deux installations du second ensemble est 2. Chacune des n villes est située à distance 1 de f_0 et à distance ε de f_{k+1}. Les distances restantes sont données par l'inégalité triangulaire. La combinaison convexe a pour coefficients $a = 1/k$ et $b = 1 - 1/k$.



Or, le coût de la combinaison convexe est $an + b\varepsilon n$. Supposons que l'algorithme sélectionne f_1 comme le plus proche voisin de f_0. L'espérance du coût des solutions produites par la procédure d'arrondi randomisé est $n(b\varepsilon + a^2 + ab(2+\varepsilon))$. Lorsque ε tend vers 0, le coût de la combinaison convexe tend vers na, alors que celui de la solution arrondie tend vers $na(1+b)$.

25.3.4 Saut intégral

L'algorithme ci-dessus majore par 6 le saut intégral de la relaxation (25.2). L'instance suivante le minore par 2, asymptotiquement.



Ce graphe est une étoile à $n+1$ sommets avec des arêtes de coût unitaire. F est l'ensemble des $n+1$ sommets, C est un ensemble de $n-1$ sommets périphériques, et $k = n-1$. Une solution entière optimale ouvre $n-1$ installations périphériques et coûte 2. Considérons la solution fractionnaire suivante : ouvrons une fraction $1/(n-1)$ de l'installation centrale et une fraction $(n-2)/(n-1)$ de chaque installation sur la périphérie. Le coût de cette solution est $n/(n-1)$, d'où le ratio $2(n-1)/n$.

25.4 Relaxation lagrangienne et algorithmes d'approximation

Cette section généralise les idées développées dans les sections précédentes. Commençons par rappeler le principe fondamental de la relaxation lagrangienne en optimisation combinatoire. Cette technique consiste à intégrer les contraintes dans la fonction objectif, pondérées par un multiplicateur de Lagrange.

Appliquons cette relaxation à la contrainte du programme entier (25.1) qui indique qu'au plus k installations peuvent être ouvertes. Notons λ le multiplicateur de Lagrange associé.

$$\begin{aligned}
&\text{minimiser} && \sum_{i\in F,\, j\in C} c_{ij}x_{ij} + \lambda\left(\sum_{i\in F} y_i - k\right) && && (25.4)\\
&\text{sous les contraintes} && \sum_{i\in F} x_{ij} \geqslant 1, && j\in C\\
& && y_i - x_{ij} \geqslant 0, && i\in F,\ j\in C\\
& && x_{ij}\in\{0,1\}, && i\in F,\ j\in C\\
& && y_i\in\{0,1\}, && i\in F
\end{aligned}$$

C'est exactement le programme en nombres entiers pour le placement d'installations, pour peu que les coûts des installations soient tous égaux (à λ). On trouve également un terme constant supplémentaire $-\lambda k$ dans la fonction objectif. Nous pouvons supposer sans perte de généralité qu'une solution optimale $(\boldsymbol{x}, \boldsymbol{y})$ du programme entier (25.1) ouvre exactement k installations. $(\boldsymbol{x}, \boldsymbol{y})$ est aussi une solution réalisable du programme entier (25.4) avec la même valeur objectif. Par conséquent, pour toute valeur de λ, la valeur du programme entier (25.4) minore celle du programme entier (25.1).

Ainsi, la relaxation lagrangienne du problème de la k-médiane est celle du placement d'installations. La contrainte globale qu'au plus k installations soient ouvertes est convertie en une pénalité par ouverture d'installations, cette pénalité étant le multiplicateur de Lagrange (voir exercice 25.4 pour un autre exemple d'application de ces idées).

L'observation importante suivante est que dans l'algorithme d'approximation pour le placement d'installations (théorème 24.7) les variables duales paient exactement le coût d'ouverture des installations, c'est-à-dire avec un facteur d'approximation 1 (voir exercice 22.9 pour d'autres algorithmes de ce type).

La dernière difficulté était de trouver la bonne valeur de λ, c'est-à-dire telle que l'algorithme de placement d'installations ouvre exactement k installations. Comme l'algorithme de sélection d'installation fonctionne avec la relaxation linéaire du problème, il a suffi de prendre la combinaison linéaire convexe de deux solutions (entières) pour trouver une solution (fractionnaire) réalisable. La dernière étape consistait à arrondir cette solution fractionnaire (spéciale) pour obtenir une solution entière. Pour le problème de la k-médiane, nous avons utilisé une procédure d'arrondi randomisé (se reporter à l'exercice 25.4 pour d'autres procédures d'arrondi).

25.5 Exercices

25.1 (Lin et Vitter [196]) Considérons la version générale du problème de la k-médiane, où les coûts de connexion ne sont pas nécessairement métriques. Proposez une réduction à partir du problème de la couverture par ensembles, qui montre qu'approcher ce problème à un facteur garanti est aussi difficile qu'approcher la couverture par ensembles, et donc que ce problème ne peut être approché à un facteur inférieur à $\Omega(\log n)$, à moins que $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log \log n)})$ (voir le chapitre 29).

25.2 Donnez le dual de la relaxation du programme (25.4) (le terme constant de la fonction objectif sera juste reporté). Quelle est sa relation avec le dual du programme pour la k-médiane ?

25.3 Appliquez la méthode de la relaxation lagrangienne pour obtenir un algorithme d'approximation à un facteur constant pour la généralisation suivante des problèmes du placement d'installations et de la k-médiane. Il s'agit du problème du placement d'installations métrique sans capacité, mais avec la contrainte supplémentaire qu'au maximum k installations peuvent être ouvertes. C'est bien une généralisation des deux problèmes : si k vaut n_f, nous obtenons le premier problème, et si les coûts d'ouverture sont nuls, nous obtenons le second.

25.4 (Garg [101] et Chudak, Roughgarden et Williamson [50]) Étudions la variante suivante du problème de l'arbre de Steiner métrique :

Problème 25.6 (k-MST métrique)[4] Étant donné un graphe complet non orienté $G = (V, E)$, un sommet distingué $r \in V$, un entier $k > 0$ et

[4] *Metric k-MST*, en anglais.

une fonction coût : $E \to \mathbf{Q}^+$ métrique, trouver un arbre couvrant de poids minimum contenant exactement k sommets, et passant par r.

Nous proposons ici une 5-approximation pour ce problème.

1. Remarquez que la relaxation lagrangienne de ce problème est le problème de l'arbre de Steiner à péage (problème 22.12) donné exercice 22.9.
2. Remarquez que la solution duale de l'algorithme d'approximation pour ce dernier problème (voir exercice 22.9) paie exactement les pénalités, c'est-à-dire avec pour facteur d'approximation 1.
3. Utilisez l'algorithme du péage pour obtenir deux arbres T_1 et T_2, pour deux valeurs de la pénalité très proches, contenant chacun k_1 et k_2 sommets respectivement, avec $k_1 < k < k_2$. Proposez une combinaison convexe de ces solutions, pondérées par α_1 et α_2.
4. Montrez qu'on peut supposer que tous les sommets de G sont à distance $\leqslant \mathrm{OPT}$ de r. Pour cela, utilisez le principe de l'élagage paramétré, introduit au chapitre 5. Introduisez un paramètre t qui sera votre hypothèse sur la longueur de l'arête la plus longue utilisée dans une solution optimale, ce qui est bien un minorant de OPT. Pour chaque valeur de t, obtenez l'instance $G(t)$ en restreignant G aux sommets à distance $\leqslant t$ de r. Lancez l'algorithme sur chacun des graphes de cette famille, et renvoyez le meilleur arbre. Considérons la procédure suivante pour arrondir la combinaison convexe. Si $\alpha_2 \geqslant 1/2$, alors coût$(T_2) \leqslant 4 \cdot \mathrm{OPT}$ et éliminer $k_2 - k$ sommets de T_2 ; sinon, dédoubler chaque arête de T_2, trouver un tour eulérien, qu'on court-circuite pour obtenir un cycle constitué uniquement des sommets de T_2 qui n'appartiennent pas à T_1 (c'est-à-dire ayant moins de $k_2 - k_1$ sommets), puis sélectionner dans ce cycle le chemin le moins cher de longueur $k - k_1 - 1$, qu'on connecte à l'aide d'une arête vers le sommet r dans T_1. L'arbre résultant a exactement k sommets. Montrez que le coût de cet arbre est $\leqslant 5 \cdot \mathrm{OPT}$.
   Indication : Utilisez que $\alpha_2 = (k - k_1)/(k_2 - k_1)$.

25.5 Appliquons la relaxation lagrangienne au programme linéaire suivant. Pour simplifier, nous n'avons pas écrit les conditions de positivité explicitement.

$$\begin{aligned} &\text{minimiser} && \mathbf{c}^T \boldsymbol{x} \\ &\text{sous les contraintes} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{25.5}$$

où $\boldsymbol{A}$ est une matrice $m \times n$. Supposons que ce programme linéaire atteint son optimum en $\boldsymbol{x} = \boldsymbol{a}$, et notons OPT la valeur objectif optimale.

Notons $\boldsymbol{y}$ le vecteur m-dimensionnel des multiplicateurs de Lagrange et reportons toutes les contraintes dans la fonction objectif :

$$\min_{\boldsymbol{x}} \left(\mathbf{c}^T \boldsymbol{x} - \boldsymbol{y}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right).$$

Quel que soit $\boldsymbol{y}$, l'expression ci-dessus minore OPT. Pour le voir, il suffit de prendre $\boldsymbol{x} = \boldsymbol{a}$. Par conséquent, l'expression suivante est aussi un minorant de OPT :

$$\max_{\boldsymbol{y}} \min_{\boldsymbol{x}} \left(\mathbf{c}^T \boldsymbol{x} - \boldsymbol{y}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\right) = \max_{\boldsymbol{y}} \left(\left(\min_{\boldsymbol{x}} (\mathbf{c}^T - \boldsymbol{y}^T \boldsymbol{A})\boldsymbol{x}\right) + \boldsymbol{y}^T \boldsymbol{b}\right).$$

Si $\boldsymbol{y}$ ne satisfait pas $\boldsymbol{A}^T \boldsymbol{y} = \mathbf{c}$, alors un choix judicieux de x montre que cette expression est arbitrairement petite, et n'a donc aucun intérêt. On n'obtient donc des minorants intéressants que si $\boldsymbol{A}^T \boldsymbol{y} = \mathbf{c}$. Nous obtenons alors le programme linéaire suivant :

$$\begin{array}{ll} \text{maximiser} & \boldsymbol{y}^T \boldsymbol{b} \\ \text{sous les contraintes} & \boldsymbol{A}^T \boldsymbol{y} = \mathbf{c} \end{array} \tag{25.6}$$

Remarquez que c'est exactement le dual du programme (25.5) ! Par conséquent, la relaxation lagrangienne d'un programme linéaire est tout simplement son dual et est donc exacte.

Donnez la relaxation lagrangienne du programme suivant :

$$\begin{array}{ll} \text{minimiser} & \mathbf{c}^T \boldsymbol{x} \\ \text{sous les contraintes} & \boldsymbol{A}\boldsymbol{x} \geqslant \boldsymbol{b} \\ & \boldsymbol{x} \geqslant 0 \end{array} \tag{25.7}$$

25.6 (Jain et Vazirani [149])[5] Étudions le problème de la classification ℓ_2^2. Étant donné un ensemble de n points $S = \{v_1, \ldots, v_n\}$ dans $\mathbf{R}^d$ et un entier $k > 0$, trouver une k-classification[6] de coût minimum, c'est-à-dire trouver k points, appelés *centres* $f_1, \ldots, f_k \in \mathbf{R}^d$, qui minimisent la somme des carré des distances euclidiennes de chaque point v_i à son centre le plus proche. Ceci définit naturellement une partition des n points en k classes. Donnez une approximation à un facteur constant pour ce problème.
Indication : Commencez par montrer qu'imposer aux centres d'appartenir à S, augmente le coût d'une solution optimale d'un facteur inférieur à 2. Appliquez la solution de l'exercice 24.6 au problème modifié.

25.7 (Korupolu, Plaxton et Rajaraman [184], et Arya et al. [17]) Pour tout ensemble S de k installations, notons coût(S) le coût total de connexion de chacune des villes à son installation la plus proche dans S. Nous appellerons *échange* l'action qui consiste à remplacer une installation de S par une installation de $\overline{S}$. Voici un algorithme naturel de recherche locale, pour la

[5] ℓ_2^2 *Clustering*, en anglais.
[6] *k-Clustering*, en anglais.

k-médiane métrique : commencer par un ensemble arbitraire S de k installations ; à chaque itération, rechercher un échange qui améliore le coût de la solution ; effectuer cet échange s'il existe, et itérer ; sinon, stop. La solution obtenue est dite *localement optimale*.

Notons $G = \{o_1, \ldots, o_k\}$ une solution optimale et $L = \{s_1, \ldots, s_k\}$ une solution localement optimale. Cet exercice démontre que $\text{coût}(L) \leqslant 5 \cdot \text{coût}(G)$, et propose un algorithme d'approximation (polynomial) à un facteur constant.

1. Notons $N_G(o)$ l'ensemble des villes connectées à l'installation $o \in G$ dans la solution optimale G. De même, notons $N_L(s)$ l'ensemble des villes connectées à $s \in L$ dans la solution localement optimale L. Nous dirons que $s \in L$ *reprend* $o \in G$ si $|N_G(o) \cap N_L(s)| > |N_G(o)|/2$. Chaque $o \in G$ n'est reprise que par au plus une installation de L. Pour simplifier, nous supposons tout d'abord que toute installation $s \in L$ reprend une unique installation de G. Quitte à réindexer les installations, s_i reprend o_i pour $1 \leqslant i \leqslant k$. En remarquant que pour tout $1 \leqslant i \leqslant k$, $\text{coût}(L + o_i - s_i) \geqslant \text{coût}(L)$, démontrez que $\text{coût}(L) \leqslant 3 \cdot \text{coût}(G)$.

   Indication : $\text{coût}(L + o_i - s_i)$ est majoré par le coût de la solution suivante : les villes de $N_L(s_i) \cup N_G(o_i)$ sont connectées comme dans la solution localement optimale. Les villes de $N_G(o_i)$ sont connectées à l'installation o_i. Celles de $N_L(s_i) - N_G(o_i)$ sont connectées aux installations de $L - s_i$ en « trois coups » de telle sorte que chaque arête utilisée par G et chaque arête utilisée par L sont utilisées une fois au plus au total.

2. Démontrez qu'en l'absence de l'hypothèse simplificatrice précédente, $\text{coût}(L) \leqslant 5 \cdot \text{coût}(G)$.

   Indication : Choisissez k échanges adéquats tels que chaque installation $o \in G$ est échangée exactement une fois et chaque installation $s \in L$ est échangée au plus deux fois.

3. Renforcez la condition d'échange pour obtenir, pour tout $\varepsilon > 0$, une $(5 + \varepsilon)$-approximation, en temps polynomial en $1/\varepsilon$ et en la taille de l'instance.

25.6 Notes

Le premier algorithme d'approximation, à un facteur $O(\log n \log\log n)$, a été proposé par Bartal [23]. Le premier algorithme à un facteur constant pour le problème de la k-médiane, à un facteur $6\frac{2}{3}$, a été conçu par Charikar, Guha, Tardos et Shmoys [41], à partir du travail de Lin et Vitter [197]. Cet algorithme utilisait l'arrondi en programmation linéaire. Les résultats présentés ici sont dus à Jain et Vazirani [149]. Le meilleur facteur d'approximation connu aujourd'hui est $3 + 2/p$, pour un temps de calcul $O(n^p)$, par Arya et al. [17]. Il s'agit d'un algorithme de recherche locale qui échange p

installations simultanément (l'algorithme avec $p = 1$ est analysé dans l'exercice 25.7).

L'instance de la section 25.3.4 est due à Jain, Mahdian et Saberi [146]. Le meilleur majorant connu pour le saut intégral de la relaxation (25.2) est 4, par Charikar et Guha [40]. Se reporter à Drineas, Kannan, Frieze, Vempala et Vinay [67], pour une 2-approximation pour le problème de la classification ℓ_2^2 (voir exercice 25.6).

26 Programmation semi-définie

Dans la deuxième partie de ce livre, nous avons vu comment obtenir de façon systématique par programmation linéaire, de bons minorants (pour un problème de minimisation) de la valeur OPT des solutions optimales de nombreux problèmes **NP**-difficiles. C'est une étape clé pour l'obtention d'algorithmes d'approximation pour un problème **NP**-difficile. Il est naturel de se demander s'il n'existe pas d'autres façons de procéder, tout aussi générales.

Nous présentons dans ce chapitre une autre classe de relaxations, la *programmation vectorielle*. Elle permet d'obtenir des relaxations pour différents problèmes **NP**-difficiles, en particulier pour des problèmes qui s'expriment sous forme de *programmes quadratiques stricts* (voir définition à la section 26.1). La programmation vectorielle est équivalente à une généralisation puissante et très étudiée de la programmation linéaire, la programmation semi-définie. Les programmes semi-définis, et donc les programmes vectoriels, se résolvent à une erreur additive $\varepsilon > 0$ près (arbitrairement petite) en temps polynomial en n et $\log(1/\varepsilon)$ via l'algorithme des ellipsoïdes (voir section 26.3).

Nous illustrons ici l'utilisation de la programmation vectorielle en construisant une 0.87856-approximation pour le problème suivant (les exercices 2.1 et 16.6 en donnaient une 1/2-approximation).

Problème 26.1 (Coupe maximum (MAX-CUT)) Étant donné un graphe non orienté $G = (V, E)$ avec des poids sur les arêtes $w : E \to \mathbf{Q}^+$, trouver une partition $(S, \overline{S})$ de V qui maximise le poids total des arêtes de la coupe, c'est-à-dire des arêtes qui ont une extrémité dans S et l'autre dans $\overline{S}$.

26.1 Programmation quadratique stricte et programmation vectorielle

Un *programme quadratique* est un problème d'optimisation (de minimisation ou de maximisation) d'une fonction quadratique dont les variables sont à valeurs entières, avec des contraintes quadratiques sur ces variables. Si tous les monômes de la fonction objectif et des contraintes sont de degré 0 ou 2, nous disons que le programme est *quadratique strict*.

Voici un programme quadratique strict pour MAX-CUT. Nous associons à chaque sommet v_i, une variable indicatrice $y_i \in \{-1, +1\}$, et nous définissons la partition $(S, \overline{S})$ ainsi : $S = \{v_i : y_i = 1\}$ et $\overline{S} = \{v_i : y_i = -1\}$.

$$\begin{array}{lll} \text{maximiser} & \dfrac{1}{2} \displaystyle\sum_{1 \leqslant i < j \leqslant n} w_{ij}(1 - y_i y_j) & \\ \text{sous les contraintes} & y_i^2 = 1, & v_i \in V \\ & y_i \in \mathbf{Z}, & v_i \in V \end{array} \tag{26.1}$$

Si v_i et v_j appartiennent à des côtés opposés, alors $y_i y_j = -1$ et la contribution de l'arête $v_i v_j$ à la fonction objectif vaut w_{ij}. Inversement, s'ils sont du même côté, $y_i y_j = 1$ et la contribution de l'arête $v_i v_j$ est nulle. Par conséquent, toute solution optimale du programme suivant est une coupe maximum de G. Relaxons ce programme en un programme vectoriel. Un *programme vectoriel* est défini sur n *variables vectorielles* dans $\mathbf{R}^n$, disons $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$, et consiste à optimiser (minimiser ou maximiser) une fonction linéaire des produits scalaires $\boldsymbol{v}_i \cdot \boldsymbol{v}_j$, pour $1 \leqslant i \leqslant j \leqslant n$, sous des contraintes linéaires sur ces produits scalaires. Un programme vectoriel peut donc être vu de façon équivalente comme un programme linéaire où chaque variable a été remplacée par le produit scalaire de deux des vecteurs.

On peut associer à tout programme quadratique strict sur n variables entières, un programme vectoriel sur n variables vectorielles de $\mathbf{R}^n$ de la façon suivante. Chacune des n variables est remplacée par une des n variables vectorielles, et les termes de degré 2 sont remplacés par les produits scalaires correspondants. Par exemple, le terme $y_i y_j$ de (26.1) est remplacé par $\boldsymbol{v}_i \cdot \boldsymbol{v}_j$. Nous associons ainsi le programme vectoriel suivant à MAX-CUT.

$$\begin{array}{lll} \text{maximiser} & \dfrac{1}{2} \displaystyle\sum_{1 \leqslant i < j \leqslant n} w_{ij}(1 - \boldsymbol{v}_i \cdot \boldsymbol{v}_j) & \\ \text{sous les contraintes} & \boldsymbol{v}_i \cdot \boldsymbol{v}_i = 1, & v_i \in V \\ & \boldsymbol{v}_i \in \mathbf{R}^n, & v_i \in V \end{array} \tag{26.2}$$

Les contraintes $\boldsymbol{v}_i \cdot \boldsymbol{v}_i = 1$ imposent que les vecteurs $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ appartiennent à la sphère n-dimensionnelle S_{n-1}. À toute solution réalisable de (26.1) correspond une solution à (26.2) ayant la même valeur objectif, en prenant $v_i = (y_i, 0, \ldots, 0)$ — remarquez que pour cette instance, $\boldsymbol{v}_i \cdot \boldsymbol{v}_j$ vaut simplement $y_i y_j$. Le programme vectoriel (26.2) est donc une relaxation du programme quadratique strict (26.1). Il s'agit clairement d'une propriété générale : tout programme vectoriel est une relaxation du programme quadratique strict associé.

Il est intéressant de noter que les programmes vectoriels sont approximables à une précision arbitraire en temps polynomial. Ainsi, la relaxation (26.2) donne un majorant de OPT pour MAX-CUT. Pour le démontrer,

rappelons quelques propriétés particulièrement puissantes des matrices semi-définies positives.

Remarque 26.2 Les programmes vectoriels ne sont pas toujours associés naturellement à des programmes quadratiques stricts. L'exercice 26.13 donne un problème **NP**-difficile qui admet une relaxation vectorielle, mais pour lequel nous ne connaissons pas de programme quadratique strict.

26.2 Matrices semi-définies positives

Soit $\boldsymbol{A}$ une matrice $n \times n$ réelle symétrique. Nous savons que $\boldsymbol{A}$ est diagonalisable (c'est-à-dire admet n vecteur propres linéairement indépendants) et que ses n valeurs propres sont réelles (mais pas nécessairement distinctes). Nous dirons que $\boldsymbol{A}$ est *semi-définie positive* si

$$\forall \boldsymbol{x} \in \mathbf{R}^n,\ \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geqslant 0.$$

Les trois conditions suivantes sont équivalentes. Elles nous seront très utiles par la suite. Nous en donnons ici une preuve schématique par souci de complétude.

Théorème 26.3 *Pour toute matrice $n \times n$ $\boldsymbol{A}$ réelle symétrique, les propositions suivantes sont équivalentes :*

1. $\forall \boldsymbol{x} \in \mathbf{R}^n,\ \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geqslant 0$.
2. *Toutes les valeurs propres de $\boldsymbol{A}$ sont positives.*
3. *Il existe une matrice $n \times n$ réelle $\boldsymbol{W}$, telle que $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W}$.*

Preuve : (1 $\Rightarrow$ 2) : Soient λ une valeur propre de $\boldsymbol{A}$ et $\boldsymbol{v}$ un vecteur propre associé. Nous avons $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$. En multipliant à gauche par $\boldsymbol{v}^T$: $\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}^T\boldsymbol{v}$. Or, d'après (1), $\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} \geqslant 0$. Ainsi, $\lambda\boldsymbol{v}^T\boldsymbol{v} \geqslant 0$. Comme $\boldsymbol{v}^T\boldsymbol{v} > 0$, $\lambda \geqslant 0$.

(2 $\Rightarrow$ 3) : Notons $\lambda_1, \ldots, \lambda_n$ les n valeurs propres de $\boldsymbol{A}$, et $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ une base orthonormale de vecteurs propres associés. Considérons la matrice $\mathbf{Q}$ ayant pour colonnes $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$, et $\boldsymbol{\Lambda}$ la matrice diagonale de coefficients $\lambda_1, \ldots, \lambda_n$. Comme pour tout i, $\boldsymbol{A}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i$, nous avons $\boldsymbol{A}\mathbf{Q} = \mathbf{Q}\boldsymbol{\Lambda}$. Comme $\mathbf{Q}$ est orthogonale, c'est-à-dire $\mathbf{Q}\mathbf{Q}^T = I$, nous avons $\mathbf{Q}^T = \mathbf{Q}^{-1}$. Ainsi,

$$\boldsymbol{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T.$$

Notons $\boldsymbol{D}$ la matrice diagonale de coefficients $\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}$ (d'après (2), $\lambda_1, \ldots, \lambda_n$ sont positives et leurs racines carrées sont donc réelles). Nous avons, $\boldsymbol{\Lambda} = \boldsymbol{D}\boldsymbol{D}^T$. Nous en concluons que :

$$\boldsymbol{A} = \mathbf{Q}\boldsymbol{D}\boldsymbol{D}^T\mathbf{Q}^T = (\mathbf{Q}\boldsymbol{D})(\mathbf{Q}\boldsymbol{D})^T.$$

Il suffit alors de poser $\boldsymbol{W} = (\boldsymbol{QD})^T$ pour conclure (3).

$(3 \Rightarrow 1)$: Pour tout $\boldsymbol{x} \in \mathbf{R}^n$,

$$\boldsymbol{x}^T\boldsymbol{Ax} = \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{Wx} = (\boldsymbol{Wx})^T(\boldsymbol{Wx}) \geqslant 0. \qquad \square$$

La décomposition de Cholesky (voir section 26.7) permet de réécrire, en temps polynomial, toute matrice réelle symétrique sous la forme $\boldsymbol{A} = \boldsymbol{U\Lambda U}^T$, où $\boldsymbol{\Lambda}$ est une matrice diagonale dont les coefficients sont les valeurs propres de $\boldsymbol{A}$. Or, $\boldsymbol{A}$ est semi-définie positive ssi tous les coefficients de $\boldsymbol{\Lambda}$ sont positifs. Nous avons donc un algorithme polynomial pour tester si une matrice est semi-définie positive. La décomposition $\boldsymbol{WW}^T$ n'est pas polynomiale en général, car elle peut contenir des coefficients irrationnels. Cependant, elle peut être approchée à une précision arbitraire en approchant les racines carrées des coefficients de $\boldsymbol{\Lambda}$. Dans la suite de ce chapitre, nous supposerons sans perte de généralité que nous connaissons la décomposition exacte, car l'erreur produite par l'approximation de la décomposition peut être intégrée dans le facteur d'approximation (voir exercice 26.6).

Remarquez enfin que la somme de deux matrices $n \times n$ semi-définies positives est semi-définie positive (en utilisant par exemple, la caractérisation (1) du théorème 26.3). Il en va de même pour toute combinaison convexe de telles matrices. L'ensemble des matrices $n \times n$ semi-définies positives forme donc un cône convexe positif.

26.3 Programmation semi-définie

Soit $\boldsymbol{Y}$ une matrice $n \times n$ dont le (i, j)-ième cœfficient est une variable réelle y_{ij}. La *programmation semi-définie* consiste à maximiser une combinaison linéaire des variables (y_{ij}) sous des contraintes linéaires sur les (y_{ij}) et en imposant que la matrice $\boldsymbol{Y}$ soit symétrique semi-définie positive.

Introduisons les notations suivantes pour poser la définition correctement. Notons $\mathbf{R}^{n\times n}$ l'espace des matrices réelles $n \times n$. Rappelons que la *trace* $\operatorname{tr} \boldsymbol{A}$ d'une matrice $\boldsymbol{A} \in \mathbf{R}^{n\times n}$ est la somme de ces coefficients diagonaux. Le produit interne de Frobenius $\boldsymbol{A} \bullet \boldsymbol{B}$ de deux matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{R}^{n\times n}$ est défini par :

$$\boldsymbol{A} \bullet \boldsymbol{B} = \operatorname{tr}(\boldsymbol{A}^T\boldsymbol{B}) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ij},$$

où a_{ij} et b_{ij} sont les (i, j)-ième coefficients de $\boldsymbol{A}$ et $\boldsymbol{B}$, respectivement. Notons M_n le cône des matrices $n \times n$ réelles symétriques. Pour tout $\boldsymbol{A} \in M_n$, nous noterons $\boldsymbol{A} \succcurlyeq 0$ pour signifier que $\boldsymbol{A}$ est semi-définie positive.

Soient $\boldsymbol{C}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_k \in M_n$ et $d_1, \ldots, d_k \in \mathbf{R}$. Un problème de programmation semi-définie s'écrit en toute généralité sous la forme du problème de maximisation suivant, que nous notons $\mathcal{S}$.

$$\begin{array}{lll} \text{maximiser} & \boldsymbol{C} \bullet \boldsymbol{Y} & (26.3) \\ \text{sous les contraintes} & \boldsymbol{D}_i \bullet \boldsymbol{Y} = d_i, \quad 1 \leqslant i \leqslant k & \\ & \boldsymbol{Y} \succcurlyeq 0, & \\ & \boldsymbol{Y} \in M_n. & \end{array}$$

Remarquez que si les matrices $\boldsymbol{C}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_k$ sont diagonales, il s'agit simplement d'un programme linéaire. Comme pour la programmation linéaire, toute inégalité linéaire dans les contraintes peut être reformulée par des égalités.

Nous dirons qu'une matrice de $\mathbf{R}^{n \times n}$ est une *solution réalisable* si elle satisfait toutes les contraintes de $\mathcal{S}$. Comme toute combinaison convexe de matrices semi-définies positives est semi-définie positive, l'espace des solutions réalisable est *convexe*, c'est-à-dire si $\boldsymbol{A} \in \mathbf{R}^{n \times n}$ et $\boldsymbol{B} \in \mathbf{R}^{n \times n}$ sont réalisables, il en va de même pour toute combinaison convexe de $\boldsymbol{A}$ et $\boldsymbol{B}$.

Soit $\boldsymbol{A} \in \mathbf{R}^{n \times n}$ une solution qui n'est pas réalisable. Soit $\boldsymbol{C} \in \mathbf{R}^{n \times n}$. Nous dirons qu'un hyperplan $\{\boldsymbol{Y} : \boldsymbol{C} \bullet \boldsymbol{Y} \leqslant b\}$ est *séparateur* pour $\boldsymbol{A}$ s'il contient l'ensemble des solutions réalisables, mais pas $\boldsymbol{A}$. Le théorème suivant démontre comment trouver un hyperplan séparateur en temps polynomial. Nous en déduisons que tout programme semi-défini peut être résolu par l'algorithme des ellipsoïdes, moyennant une erreur additive $\varepsilon > 0$ arbitrairement petite, en temps polynomial en n et $\log(1/\varepsilon)$ (se reporter à la section 26.7 pour d'autres méthodes plus efficaces).

Théorème 26.4 *Soient $\mathcal{S}$ un programme semi-défini et $\boldsymbol{A}$ un point de $\mathbf{R}^{n \times n}$. Nous pouvons décider en temps polynomial si $\boldsymbol{A}$ est réalisable ou non, et si ce n'est pas le cas, trouver un hyperplan séparateur.*

Preuve : Déterminer si $\boldsymbol{A}$ est réalisable consiste à tester si $\boldsymbol{A}$ est symétrique et semi-définie positive et si elle vérifie toutes les contraintes linéaires. Suite aux remarques de la section 26.2, cela se fait en temps polynomial. Si $\boldsymbol{A}$ n'est pas réalisable, on construit un hyperplan séparateur ainsi :

- Si $\boldsymbol{A}$ n'est pas symétrique, il existe i et j tels que $a_{ij} > a_{ji}$. Alors $\{\boldsymbol{Y} : y_{ij} \leqslant y_{ji}\}$ est un hyperplan séparateur.
- Si $\boldsymbol{A}$ n'est pas semi-définie positive, alors elle admet une valeur propre strictement négative λ. Notons $\boldsymbol{v}$ le vecteur propre correspondant. Alors, $(\boldsymbol{v}\boldsymbol{v}^T) \bullet \boldsymbol{Y} = \boldsymbol{v}^T \boldsymbol{Y} \boldsymbol{v} \geqslant 0$ définit un hyperplan séparateur.
- Enfin, si n'importe laquelle des contraintes linéaires est violée, celle-ci définit directement un hyperplan séparateur.

□

Nous allons maintenant démontrer que les programmes vectoriels sont équivalents aux programmes semi-définis, démontrant ainsi qu'ils peuvent se résoudre efficacement avec une précision arbitraire. Notons $\mathcal{V}$ un programme vectoriel à n variables vectorielles $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ de dimension n. Nous lui associons le programme semi-défini $\mathcal{S}$ suivant à n^2 variables $(y_{ij})_{1 \leqslant i,j \leqslant n}$. Nous

obtenons $\mathcal{S}$ en remplaçant chaque produit scalaire $\boldsymbol{v}_i \cdot \boldsymbol{v}_j$ de $\mathcal{V}$ par la variable y_{ij}. La fonction objectif et les contraintes sont maintenant linéaires en les (y_{ij}). Nous imposons enfin que la matrice $\boldsymbol{Y} = (y_{ij})$ soit symétrique et semi-définie positive.

Lemme 26.5 *Le programme vectoriel $\mathcal{V}$ est équivalent au programme semi-défini $\mathcal{S}$.*

Preuve : Démontrons qu'à toute solution réalisable de $\mathcal{V}$, correspond une solution réalisable de $\mathcal{S}$ de même valeur objectif, et vice-versa. Soit $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ une solution réalisable de $\mathcal{V}$. Notons W la matrice dont les colonnes sont $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$. Par construction, $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W}$ est une solution réalisable de $\mathcal{S}$ ayant la même valeur objectif.

Réciproquement, soit $\boldsymbol{A}$ une solution réalisable de $\mathcal{S}$. D'après le théorème 26.3, il existe une matrice $n \times n$ W, telle que $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W}$. Notons $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ les colonnes de $\boldsymbol{W}$. Alors, par construction, $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ est bien une solution réalisable de $\mathcal{V}$ ayant la même valeur objectif. □

Nous obtenons donc la relaxation semi-définie suivante pour MAX-CUT, équivalente au programme vectoriel (26.2).

$$\begin{aligned} &\text{maximiser} && \frac{1}{2}\sum_{1 \leqslant i < j \leqslant n} w_{ij}(1 - y_i y_j) && && (26.4)\\ &\text{sous les contraintes} && y_i^2 = 1, && v_i \in V\\ & && \boldsymbol{Y} \succcurlyeq 0,\\ & && \boldsymbol{Y} \in M_n. \end{aligned}$$

26.4 Approximation par arrondi randomisé

Donnons à présent l'algorithme pour MAX-CUT. Pour simplifier, nous supposons que nous avons une solution optimale du programme vectoriel (26.2). En effet, l'erreur très faible lors de la résolution peut être intégrée dans le facteur d'approximation (voir exercice 26.6). Soit $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ une solution optimale, et OPT_v sa valeur objectif. Ces vecteurs appartiennent à la sphère unitaire S_{n-1} de $\mathbf{R}^n$. Nous recherchons une coupe de poids au moins égal à une fraction de OPT_v.

Notons θ_{ij} l'angle entre les vecteurs $\boldsymbol{a}_i$ et $\boldsymbol{a}_j$. La contribution de cette paire de vecteurs à OPT_v est :

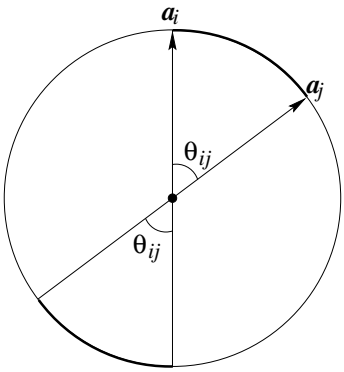
$$\frac{w_{ij}}{2}(1 - \cos\theta_{ij}).$$

Plus θ_{ij} est proche de π, plus la contribution est grande. Par conséquent, nous souhaitons séparer les sommets v_i et v_j lorsque θ_{ij} est grand. Nous

allons procéder ainsi : sélectionner un vecteur $\boldsymbol{r}$ uniformément sur la sphère unitaire S_{n-1}, et poser $S = \{v_i : \boldsymbol{a}_i \cdot \boldsymbol{r} \geqslant 0\}$.

Lemme 26.6 $\mathbf{Pr}[v_i \textit{ et } v_j \textit{ sont séparés}] = \dfrac{\theta_{ij}}{\pi}$.

Preuve : Projetons $\boldsymbol{r}$ sur le plan défini par $\boldsymbol{a}_i$ et $\boldsymbol{a}_j$. Les sommets v_i et v_j seront séparés si la perpendiculaire à cette projection appartient à un des deux arcs d'angle θ_{ij} illustrés ci-dessous.



Or, comme $\boldsymbol{r}$ est tiré suivant une distribution à symétrie sphérique, sa projection, et donc sa perpendiculaire, sont des directions aléatoires uniformes dans ce plan. CQFD. □

Le lemme suivant permet de tirer un vecteur uniformément sur la sphère S_{n-1}.

Lemme 26.7 *Soient $x_1, \ldots, x_n$ des réels tirés indépendamment suivant une loi normale de moyenne 0 et de variance 1. Poser $d = (x_1^2 + \ldots + x_n^2)^{1/2}$. Alors, $(x_1/d, \ldots, x_n/d)$ suit une loi uniforme sur la sphère unitaire S_{n-1}.*

Preuve : Considérons le vecteur $\boldsymbol{r} = (x_1, \ldots, x_n)$. La distribution de $\boldsymbol{r}$ a pour fonction de densité :

$$f(y_1, \ldots, y_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-y_i^2/2} = \frac{1}{(2\pi)^{n/2}} e^{-\frac{1}{2}\sum_i y_i^2}.$$

Remarquez que cette fonction ne dépend que de la norme du vecteur. Par conséquent, $\boldsymbol{r}$ suit une loi à symétrie sphérique. Ainsi, en divisant par la norme de $\boldsymbol{r}$, c'est-à-dire d, nous obtenons bien un vecteur aléatoire uniforme sur S_{n-1}. □

Voici donc l'algorithme.

Algorithme 26.8 (MAX-CUT)

1. Résoudre le programme vectoriel (26.2). Soit $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ une solution optimale.
2. Tirer un vecteur $\boldsymbol{r}$ aléatoire uniforme de la sphère unitaire S_{n-1}.
3. Poser $S = \{v_i : \boldsymbol{a}_i \cdot \boldsymbol{r} \geqslant 0\}$.

Notons W la variable aléatoire associée au poids total des arêtes de la coupe construite par l'algorithme 26.8, et posons

$$\alpha = \frac{2}{\pi} \min_{0 \leqslant \theta \leqslant \pi} \frac{\theta}{1 - \cos\theta}.$$

On peut démontrer que $\alpha > 0.87856$ (voir exercice 26.3).

Lemme 26.9 $\mathbf{E}[W] \geqslant \alpha \cdot \mathrm{OPT}_v$.

Preuve : Par définition de α, pour tout $0 \leqslant \theta \leqslant \pi$,

$$\frac{\theta}{\pi} \geqslant \alpha \left(\frac{1 - \cos\theta}{2} \right). \tag{26.5}$$

Ainsi, d'après le lemme 26.6,

$$\begin{aligned} \mathbf{E}[W] &= \sum_{1 \leqslant i < j \leqslant n} w_{ij} \mathbf{Pr}[\boldsymbol{a}_i \text{ et } \boldsymbol{a}_j \text{ sont séparés}] \\ &= \sum_{1 \leqslant i < j \leqslant n} w_{ij} \frac{\theta_{ij}}{\pi} \geqslant \alpha \cdot \sum_{1 \leqslant i < j \leqslant n} \frac{1}{2} w_{ij} (1 - \cos\theta_{ij}) = \alpha \cdot \mathrm{OPT}_v \,. \end{aligned}$$

□

Définissons le *saut intégral* de la relaxation (26.2) comme

$$\inf_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_v(I)},$$

où la borne inférieure est prise sur toutes les instances I de MAX-CUT.

Corollaire 26.10 *Le saut intégral de la relaxation (26.2) est supérieur à* $\alpha > 0.87856$.

Théorème 26.11 *Il existe une* 0.87856*-approximation randomisée pour MAX-CUT.*

Preuve : Commençons par obtenir une borne avec « forte probabilité » à partir du lemme 26.9. Notons T le poids total des arêtes de G, et soit a tel que $\mathbf{E}[W] = aT$. Fixons une constante $\varepsilon > 0$, et posons

$$p = \mathbf{Pr}[W < (1-\varepsilon)aT].$$

Comme la valeur de W est toujours inférieure à T,

$$aT \leqslant p(1-\varepsilon)aT + (1-p)T.$$

Ainsi,

$$p \leqslant \frac{1-a}{1-a+a\varepsilon}.$$

Or,

$$T \geqslant \mathbf{E}[W] = aT \geqslant \alpha \cdot \mathrm{OPT}_v \geqslant \alpha \cdot \mathrm{OPT} \geqslant \frac{\alpha T}{2}.$$

Cette dernière inégalité est une conséquence du fait que $\mathrm{OPT} \geqslant T/2$ (voir l'exercice 2.1). Ainsi, $\alpha/2 \leqslant a \leqslant 1$. En reportant cet encadrement de a, nous obtenons :

$$p \leqslant 1 - \frac{\varepsilon\alpha/2}{1+\varepsilon\alpha/2-\alpha/2} \leqslant 1-c,$$

où

$$c = \frac{\varepsilon\alpha/2}{1+\varepsilon\alpha/2-\alpha/2}.$$

Renvoyons-donc la coupe la plus lourde construite après $1/c$ exécutions indépendantes de l'algorithme 26.8. Notons W' le poids de cette coupe. Alors,

$$\mathbf{Pr}[W' \geqslant (1-\varepsilon)aT] \geqslant 1-(1-c)^{1/c} \geqslant 1-\frac{1}{e}.$$

Enfin, comme $aT \geqslant \alpha \cdot \mathrm{OPT} > 0.87856\ \mathrm{OPT}$, nous pouvons choisir $\varepsilon > 0$ tel que $(1-\varepsilon)aT \geqslant 0.87856\ \mathrm{OPT}$. □

Exemple 26.12 L'instance suivante démontre que la borne sur le saut intégral de la relaxation (26.2) donnée au corollaire 26.10 est presque exacte. Considérons le graphe réduit à un cycle de longueur 5 $(v_1, v_2, v_3, v_4, v_5, v_1)$. Une solution optimale de la relaxation (26.2) place les cinq vecteurs dans un sous-espace de dimension 2, dans lequel leurs positions relatives sont données par $\boldsymbol{a}_i = (\cos(\frac{4i\pi}{5}), \sin(\frac{4i\pi}{5}))$, pour $1 \leqslant i \leqslant 5$ (voir exercice 26.5). Cette solution coûte $\mathrm{OPT}_v = \frac{5}{2}(1+\cos\frac{\pi}{5}) = \frac{25+5\sqrt{5}}{8}$. Or, $\mathrm{OPT} = 4$ pour ce graphe. Le saut intégral pour cette instance est donc $\frac{32}{25+5\sqrt{5}} = 0.88445\ldots$ □

26.5 Améliorer la garantie pour MAX-2SAT

MAX-2SAT est la restriction de MAX-SAT (problème 16.1) aux formules dont les clauses contiennent au plus deux littéraux. Nous avons obtenu une 3/4-approximation randomisée pour ce problème au chapitre 16, qu'on dérandomise par la méthode de l'espérance conditionnelle. Nous proposons ici une amélioration à l'aide de la programmation semi-définie.

La difficulté est de convertir le programme quadratique évident pour ce problème (voir exercice 26.8) en un programme quadratique strict. Voici comment nous procédons. Nous associons à chaque variable booléenne x_i, une variable $y_i \in \{-1, +1\}$, pour $1 \leqslant i \leqslant n$. Introduisons une variable supplémentaire $y_0 \in \{-1, +1\}$. Considérons que la valeur de la variable booléenne x_i est vraie si $y_i = y_0$ et fausse sinon. Avec cette convention, nous pouvons écrire la valeur d'une clause en fonction des (y_i), la *valeur* $v(C)$ de la clause C étant 1 si C est satisfaite et 0 sinon. Pour les clauses composées d'un unique littéral, nous avons

$$v(x_i) = \frac{1 + y_0 y_i}{2} \text{ et } v(\overline{x_i}) = \frac{1 - y_0 y_i}{2}.$$

Pour les clauses composées de deux littéraux, par exemple $(x_i \vee x_j)$, nous obtenons

$$\begin{aligned} v(x_i \vee x_j) &= 1 - v(\overline{x_i})v(\overline{x_j}) = 1 - \frac{1 - y_0 y_i}{2} \cdot \frac{1 - y_0 y_j}{2} \\ &= \frac{1}{4}\left(3 + y_0 y_i + y_0 y_j - y_0^2 y_i y_j\right) \\ &= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_j}{4}. \end{aligned}$$

Remarquons que nous avons utilisé le fait que $y_0^2 = 1$. On peut facilement vérifier que dans tous les autres cas, la valeur d'une clause de deux littéraux est aussi une combinaison linéaire de termes de la forme $(1+y_i y_j)$ ou $(1-y_i y_j)$. Par conséquent, toute instance de MAX-2SAT se réécrit sous la forme d'un programme quadratique strict, dont les coefficients (a_{ij}) et (b_{ij}) sont obtenus en sommant ces termes.

$$\begin{array}{lll} \text{maximiser} & \displaystyle\sum_{0 \leqslant i < j \leqslant n} a_{ij}(1 + y_i y_j) + b_{ij}(1 - y_i y_j) & (26.6) \\ \text{sous les contraintes} & y_i^2 = 1, \qquad 0 \leqslant i \leqslant n & \\ & y_i \in \mathbf{Z}, \qquad 0 \leqslant i \leqslant n & \end{array}$$

Nous obtenons la relaxation suivante sous forme d'un programme vectoriel, où la variable vectorielle $\boldsymbol{v}_i$ correspond à y_i.

$$\begin{array}{lll} \text{maximiser} & \displaystyle\sum_{0\leqslant i<j\leqslant n} a_{ij}(1+\boldsymbol{v}_i\cdot\boldsymbol{v}_j)+b_{ij}(1-\boldsymbol{v}_i\cdot\boldsymbol{v}_j) & (26.7)\\ \text{sous les contraintes} & \boldsymbol{v}_i\cdot\boldsymbol{v}_i=1, \qquad 0\leqslant i\leqslant n & \\ & \boldsymbol{v}_i\in\mathbf{R}^{n+1}, \qquad 0\leqslant i\leqslant n & \end{array}$$

L'algorithme est similaire à celui pour MAX-CUT. Commencer par résoudre le programme vectoriel (26.7), et noter $\boldsymbol{a}_0,\ldots,\boldsymbol{a}_n$ une solution optimale. Puis tirer un vecteur aléatoire $\boldsymbol{r}$ uniformément sur la sphère unité S_n de $\mathbf{R}^{n+1}$, et poser $y_i=1$ ssi $\boldsymbol{r}\cdot\boldsymbol{a}_i\geqslant 0$, pour $0\leqslant i\leqslant n$. Nous obtenons ainsi une instanciation des variables booléennes. Notons W la variable aléatoire associée au poids de cette instanciation.

Lemme 26.13 $\mathbf{E}[W]\geqslant\alpha\cdot\mathrm{OPT}_v$.

Preuve :

$$\mathbf{E}[W]=2\sum_{0\leqslant i<j\leqslant n} a_{ij}\mathbf{Pr}[y_i=y_j]+b_{ij}\mathbf{Pr}[y_i\neq y_j].$$

Notons θ_{ij} l'angle entre $\boldsymbol{a}_i$ et $\boldsymbol{a}_j$. D'après l'inégalité (26.5),

$$\mathbf{Pr}[y_i\neq y_j]=\frac{\theta_{ij}}{\pi}\geqslant\frac{\alpha}{2}(1-\cos\theta_{ij}).$$

Or, l'exercice 26.4 démontre que

$$\mathbf{Pr}[y_i=y_j]=1-\frac{\theta_{ij}}{\pi}\geqslant\frac{\alpha}{2}(1+\cos\theta_{ij}).$$

Ainsi,

$$\mathbf{E}[W]\geqslant\alpha\cdot\sum_{0\leqslant i<j\leqslant n} a_{ij}(1+\cos\theta_{ij})+b_{ij}(1-\cos\theta_{ij})=\alpha\cdot\mathrm{OPT}_v\,.$$

□

26.6 Exercices

26.1 La matrice $\boldsymbol{W}$ du théorème 26.3 est-elle unique (aux changements de signe près) ?
Indication : Étudiez la matrice $\mathbf{Q}\boldsymbol{D}\mathbf{Q}^T$.

26.2 Notons $\boldsymbol{B}$ une matrice obtenue à partir d'une matrice $\boldsymbol{A}$ en supprimant un ensemble de colonnes *et* l'ensemble des lignes correspondantes. Une telle matrice $\boldsymbol{B}$ est appelée *sous-matrice principale* de la matrice $\boldsymbol{A}$.

Démontrez qu'une matrice $\boldsymbol{A}$ réelle et symétrique est semi-définie positive ssi les déterminants de ses sous-matrices principales sont tous positifs (ou nuls) (voir théorème 26.3 pour d'autres conditions).

26.3 Démontrez, par de l'analyse élémentaire, que $\alpha > 0.87856$.

26.4 Démontrez que pour tout $0 \leqslant \phi \leqslant \pi$,

$$1 - \frac{\phi}{\pi} \geqslant \frac{\alpha}{2}(1 + \cos\phi).$$

Indication : Utilisez $\theta = \pi - \phi$ dans l'inégalité (26.5).

26.5 Démontrez que pour le cycle de longueur 5, la solution donnée par l'exemple 26.12 est bien une solution optimale du programme vectoriel relâché pour MAX-CUT.

26.6 Montrez que les erreurs induites par les deux faits que la solution du programme vectoriel (26.2) ne soit pas optimale et que la matrice $\boldsymbol{A}$ n'est pas exactement $\boldsymbol{W}\boldsymbol{W}^T$ (voir la fin de la section 26.2), peuvent être absorbées dans le facteur d'approximation de MAX-CUT.
Indication : Utilisez l'astuce du théorème 26.11 et le fait que le poids de la solution du programme (26.2) appartient à $[T/2, T]$, où T est le poids total des arêtes du graphe G.

26.7 Le théorème 26.11 montre comment obtenir une formulation avec « forte probabilité » à partir du lemme 26.9. Obtenez le même résultat pour MAX-2SAT à partir du lemme 26.13, et déduisez-en une 0.87856-approximation pour MAX-2SAT.

26.8 Exhibez un programme quadratique pour MAX-2SAT.

26.9 (Linial, London, et Rabinovich [198]) Soit $G = (V, E)$, une clique à n sommets, munie d'une fonction de poids w positive sur ses arêtes. Le but est de trouver un plongement ℓ_2^2 de distorsion optimale des sommets de G. Notons $\boldsymbol{v}_i \in \mathbf{R}^n$ l'image du sommet i par un tel plongement. Les contraintes sur le plongement sont que :

1. aucune arête n'est étirée, c'est-à-dire pour tout $1 \leqslant i < j \leqslant n$, $||\boldsymbol{v}_i - \boldsymbol{v}_j||^2 \leqslant w_{ij}$, et que
2. la compression maximum est minimum, c'est-à-dire

$$\text{maximiser} \min_{(i,j)\,:\,w_{ij}\neq 0} (||\boldsymbol{v}_i - \boldsymbol{v}_j||^2 / w_{ij}).$$

Exhibez un programme vectoriel calculant un tel plongement optimal et donnez-en un programme semi-défini équivalent.

Indication : Le programme vectoriel est :

$$\begin{array}{lll} \text{minimiser} & c & (26.8) \\ \text{sous les contraintes} & \boldsymbol{v}_i \cdot \boldsymbol{v}_i + \boldsymbol{v}_j \cdot \boldsymbol{v}_j - 2\boldsymbol{v}_i \cdot \boldsymbol{v}_j \leqslant w_{ij}, & 1 \leqslant i < j \leqslant n \\ & \boldsymbol{v}_i \cdot \boldsymbol{v}_i + \boldsymbol{v}_j \cdot \boldsymbol{v}_j - 2\boldsymbol{v}_i \cdot \boldsymbol{v}_j \geqslant cw_{ij}, & 1 \leqslant i < j \leqslant n \\ & \boldsymbol{v}_i \in \mathbf{R}^n, & 1 \leqslant i \leqslant n \end{array}$$

26.10 (Knuth [182]) Proposez un algorithme efficace pour tirer un réel suivant une loi normale de moyenne 0 et de variance 1, à partir d'une source de bits aléatoires non biaisés.

26.11 Proposez un programme quadratique strict pour les problèmes MAX k-CUT et la coupe maximum orientée (problèmes 2.14 et 2.15 énoncés aux exercices 2.3 et 2.4). Donnez-en une relaxation sous forme d'un programme vectoriel puis d'un programme semi-défini équivalent.

26.12 (Goemans et Williamson [113]) Étudions MAX-CUT avec la contrainte supplémentaire qu'on impose à des paires distinguées de sommets d'être d'un même côté ou d'être séparées par la coupe. Formellement, deux ensembles de paires de sommets sont données : S_1, les paires qui doivent être séparées, et S_2, les paires de sommets qui doivent être placées du même côté de la coupe. Il s'agit de trouver une coupe de poids maximum sous ces nouvelles contraintes. Nous supposons que les contraintes induites par S_1 et S_2 sont consistantes. Exhibez un programme quadratique strict, puis une relaxation sous forme d'un programme vectoriel pour ce problème. Adaptez l'algorithme 26.8 pour obtenir le même facteur d'approximation.

26.13 (Karger, Motwani, et Sudan [166]) Soit $G = (V, E)$ un graphe non orienté. Considérons le programme vectoriel à n variables vectorielles de dimension n, une pour chaque sommet de G, avec pour contraintes que les vecteurs appartiennent à la sphère unité S_{n-1} et que pour chaque arête $ij \in E$,

$$\boldsymbol{v}_i \cdot \boldsymbol{v}_j \leqslant -\frac{1}{k-1}.$$

Démontrez que ce programme vectoriel est une relaxation du problème de la k-coloration, c'est-à-dire que si G est k-coloriable, alors ce programme vectoriel admet une solution réalisable.
Indication : Considérer les k vecteurs suivants de $\mathbf{R}^n$. Leurs $n-k$ dernières coordonnées sont nulles. La j-ième coordonnée du vecteur i est $-\sqrt{\frac{k-1}{k}}$ pour $j = i$ et $1/\sqrt{k(k-1)}$ sinon.

26.14 (Chor et Sudan [46]) Étudions le problème suivant :

Problème 26.14 (Entre-deux)[1] Considérons un ensemble de n objets $S = \{x_1, x_2, \ldots, x_n\}$ et un ensemble $T \subseteq S \times S \times S$ de m triplets, où les trois objets de chaque triplet sont toujours distincts. Nous dirons qu'un ordre total (une permutation) de S, $x_{\pi_1} < x_{\pi_2} < \cdots < x_{\pi_n}$, *satisfait* le triplet $(x_i, x_j, x_k) \in T$ si x_j est placé entre x_i et x_k dans l'ordre, c'est-à-dire si $x_i < x_j < x_k$ ou $x_k < x_j < x_i$. Le problème est de trouver un ordre total qui maximise le nombre de triplets satisfaits.

1. Démontrer qu'un ordre aléatoire (c'est-à-dire une permutation aléatoire tirée uniformément parmi toutes les permutations) satisfait un tiers des triplets de T en moyenne.
2. Appliquer la méthode de l'espérance conditionnelle pour dérandomiser l'algorithme ci-dessus et obtenir une 1/3-approximation. Quel est le majorant de OPT utilisé dans l'analyse de cet algorithme ? Démontrez sur un exemple qu'aucun algorithme ne peut faire mieux, comparé à ce majorant.
3. La suite de cet exercice propose un algorithme fondé sur la programmation semi-définie. Les idées s'expliquent plus simplement en supposant que l'instance est satisfaisable, c'est-à-dire qu'il existe un ordre qui satisfait les m triplets simultanément. Remarquez que tester la satisfaisabilité est **NP**-difficile, par conséquent la restriction du problème de l'entre-deux à ces instances n'est pas un problème d'optimisation **NP** (voir exercice 1.9). Démontrez qu'une instance est satisfaisable ssi le programme quadratique strict suivant, de variables $p_i \in \mathbf{R}$, $i = 1, \ldots, n$, admet une solution :

$$\begin{aligned} (p_i - p_j)^2 &\geqslant 1 \quad \text{pour tout } i, j, \\ (p_i - p_j)(p_k - p_j) &\leqslant 0 \quad \text{pour tout } (x_i, x_j, x_k) \in T. \end{aligned}$$

4. Donnez le programme vectoriel relâché associé à ce programme quadratique strict, ainsi que son programme semi-défini équivalent.
5. Exhibez une instance dont le programme semi-défini associé ci-dessus est satisfaisable, mais pas l'instance elle-même.
6. Supposons qu'une matrice $n \times n$ Y soit une solution réalisable du programme semi-défini ci-dessus, et considérons $v_i \in \mathbf{R}^n$, $i = 1, \ldots, n$, les vecteurs tels que $Y_{ij} = v_i^T v_j$. Tirez uniformément un vecteur r au hasard sur la sphère unité S_{n-1}. Considérez l'ordre aléatoire obtenu en triant les $r^T v_i$. Démontrez que cet ordre satisfait au moins la moitié des triplets de T en moyenne.

   Indication : Quelle est la probabilité qu'un triplet donné soit satisfait ? Quel est l'angle entre $v_i - v_j$ et $v_k - v_j$?

[1] *Betweenness*, en anglais.

26.7 Notes

Les résultats de ce chapitre reposent sur le travail majeur de Goemans et Williamson [113] qui introduisirent la programmation semi-définie en algorithmique d'approximation. Les résultats expérimentaux présentés dans leur article montrent que le facteur d'approximation de l'algorithme 26.8 est bien meilleur sur les instances typiques, que dans le pire cas. Mahajan et Ramesh [208] dérandomisèrent l'algorithme 26.8, ainsi que l'algorithme pour MAX-2SAT, à l'aide de la méthode de l'espérance conditionnelle. Karloff [169] exhiba une famille d'instances critiques pour l'algorithme 26.8, où le poids de la coupe générée est arbitrairement proche de $\alpha \cdot \mathrm{OPT}_v$ en moyenne. Feige et Schechtman [91] renforcèrent ce résultat en démontrant qu'il existe des graphes tels que même le meilleur hyperplan (plutôt que l'hyperplan aléatoire de l'algorithme 26.8) produit une coupe de poids $\alpha \cdot \mathrm{OPT}_v$ seulement. Ils démontrèrent également que le saut intégral de la relaxation semi-définie (26.2) pour MAX-CUT vaut α.

Reportez-vous à Alizadeh [6], Nesterov et Nemirovskii [223] et Overton [224] pour des algorithmes efficaces de résolution approchée des programmes semi-définis, utilisant la méthode des points intérieurs. Référez-vous à Wolkowitz [269] et Vandenberghe et Boyd [259] pour la théorie de la dualité en programmation semi-définie.

Lovász et Schrijver [204] utilisèrent la programmation semi-définie pour renforcer automatiquement toute relaxation convexe (c'est-à-dire dont le domaine des solutions réalisables est convexe) d'un programme entier à valeurs dans $\{0, 1\}$. Ils démontrèrent également que si la relaxation initiale se résout en temps polynomial, il en va de même pour la relaxation renforcée (notez que pour garantir un temps polynomial, on ne peut itérer ce processus qu'un nombre constant de fois).

Feige et Goemans [88] améliorèrent le facteur d'approximation pour MAX-2SAT à 0.931. Ils proposèrent également une 0.859-approximation pour le problème de la coupe orientée maximum (voir exercice 26.11). Reportez-vous à Frieze et Jerrum [97] pour des algorithmes à base de programmation semi-définie pour le problème MAX k-CUT. Karger, Motwani, et Sudan [166] utilisèrent la relaxation de l'exercice 26.13 pour construire une $O(n^{1-3/(k+1)} \log^{1/2} n)$-coloration pour tout graphe k-coloriable.

Troisième partie

Autres sujets d'étude

27 Vecteur le plus court

Le problème du vecteur le plus court est au cœur des aspects calculatoires de la géométrie des nombres. L'algorithme d'approximation présenté ici a de nombreuses applications en théorie des nombres et en cryptographie. Deux des plus importantes applications sont l'obtention d'algorithmes polynomiaux pour factoriser les polynômes sur le corps des rationnels et l'approximation diophantienne simultanée.

Problème 27.1 (Vecteur le plus court)[1] Étant donné n vecteurs linéairement indépendants $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n \in \mathbf{Q}^n$, trouver le vecteur (non nul) le plus court, pour la norme euclidienne, du module[2] engendré par ces vecteurs. Le module $\mathcal{L}$ engendré par $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ est l'ensemble des combinaisons linéaires entières de ces vecteurs, c'est-à-dire $\mathcal{L} = \{\lambda_1 \boldsymbol{a}_1 + \cdots + \lambda_n \boldsymbol{a}_n : \lambda_i \in \mathbf{Z}\}$.

Remarque 27.2 Nous ne considérerons que des modules de rang total, c'est-à-dire des modules qui remplissent entièrement l'espace sur lequel ils sont définis. La plupart des résultats s'étendent au cas général, mais les preuves sont plus compliquées.

Nous allons présenter un algorithme ayant un facteur d'approximation exponentiel (en n) pour ce problème. La recherche d'un algorithme ayant un facteur polynomial est un problème ouvert depuis plus d'une vingtaine d'années. Néanmoins, il est important de remarquer que cet algorithme de facteur exponentiel est très efficace et très utilisé en pratique.

Le vecteur le plus court d'un module unidimensionnel engendré par deux entiers est tout simplement le plus grand diviseur commun (pgcd)[3] des deux entiers, qui se calcule en temps polynomial avec l'algorithme d'Euclide. En dimension 2, le problème du vecteur le plus court se résout aussi en temps polynomial. C'est une conséquence de l'algorithme de Gauss formulé initialement dans le langage des formes quadratiques. Il sera instructif de commencer par étudier ces deux algorithmes, puisque l'algorithme de Gauss est en quelque sorte une généralisation de celui d'Euclide et que l'algorithme n-dimensionnel est une généralisation de celui de Gauss.

Le problème du vecteur le plus court tient une place particulière dans ce livre pour plusieurs raisons. Contrairement aux autres problèmes **NP**-

[1] *Shortest vector*, en anglais.
[2] $\mathbb{Z}$-espace vectoriel ; *Lattice*, en anglais.
[3] *Greatest common divisor (gcd)*, en anglais.

difficiles étudiés dans ce livre, nous ne savons pas s'il existe une instance (un module) qui admet un nombre exponentiel de plus courts vecteurs. Nous ne savons pas non plus si ce problème est **NP**-difficile au sens usuel ; nous savons uniquement qu'il est **NP**-difficile par réduction randomisée. Une autre particularité est que la technique de minoration utilisée pour l'algorithme de facteur exponentiel pourrait, en principe, donner lieu à une approximation d'un facteur polynomial – nous démontrerons par une preuve existentielle (non constructive) qu'elle permet en effet d'obtenir un certificat négatif approché pour le problème du vecteur le plus court. Ce problème est donc différents des autres problèmes étudiés dans ce livre, où le meilleur minorant connu est toujours calculable en temps polynomial.

27.1 Bases, déterminants et défaut d'orthogonalité

Tous les vecteurs de ce chapitre sont des vecteurs ligne, sauf spécification contraire. Nous noterons $\boldsymbol{A}$ la matrice $n \times n$ dont les lignes sont les vecteurs $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ donnés en entrée. Soient $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathcal{L}$ des vecteurs, et $\boldsymbol{B}$ la matrice $n \times n$ dont les lignes sont $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$. Puisque $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ engendrent $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, $\boldsymbol{B} = \boldsymbol{\Lambda}\boldsymbol{A}$, où $\boldsymbol{\Lambda}$ est une matrice $n \times n$ à coefficients entiers. Ainsi, $\det(\boldsymbol{B})$ est un multiple entier de $\det(\boldsymbol{A})$. Nous dirons que $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ forment une *base* du module $\mathcal{L}$ si le module engendré par ces vecteurs est exactement $\mathcal{L}$. Nous dirons qu'une matrice carrée à coefficients entiers est *unimodulaire* si son déterminant est ± 1. Remarquons que l'inverse d'une matrice unimodulaire est aussi unimodulaire.

Théorème 27.3 *Pour tous vecteurs $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathcal{L}$, les propositions suivantes sont équivalentes :*

1. *$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ forment une base du module $\mathcal{L}$.*
2. *$|\det(\boldsymbol{B})| = |\det(\boldsymbol{A})|$.*
3. *il existe une matrice $n \times n$ unimodulaire $\boldsymbol{U}$ telle que $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{A}$.*

Preuve : Puisque $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ engendrent $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, $\boldsymbol{B} = \boldsymbol{\Lambda}\boldsymbol{A}$, où $\boldsymbol{\Lambda}$ est une matrice $n \times n$ à coefficients entiers.

$1 \Rightarrow 2$: Si $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ est une base de $\mathcal{L}$, ils engendrent $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$. Donc, $\boldsymbol{A} = \boldsymbol{\Lambda}'\boldsymbol{B}$, où $\boldsymbol{\Lambda}'$ est une matrice $n \times n$ à coefficients entiers. Ainsi, $\det(\boldsymbol{\Lambda}) \det(\boldsymbol{\Lambda}') = 1$. Or, les déterminants de $\boldsymbol{\Lambda}$ et $\boldsymbol{\Lambda}'$ sont des entiers, et donc $\det(\boldsymbol{\Lambda}) = \pm 1$, et $|\det(\boldsymbol{B})| = |\det(\boldsymbol{A})|$.

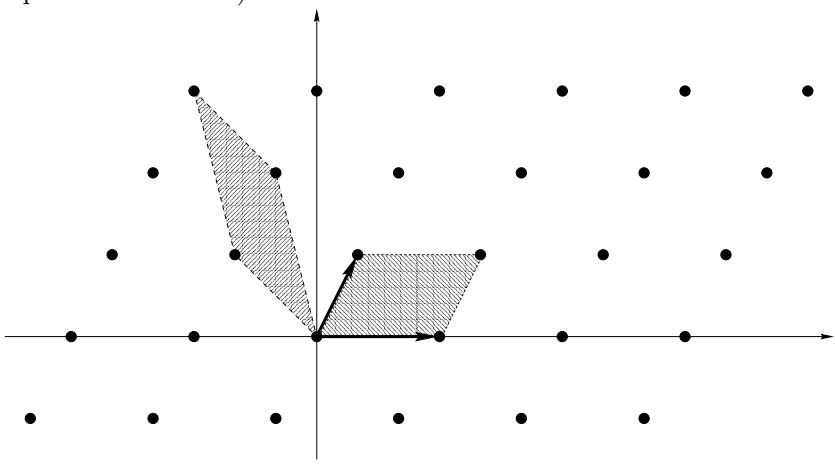
$2 \Rightarrow 3$: Comme $|\det(\boldsymbol{B})| = |\det(\boldsymbol{A})|$, nous avons $\det(\boldsymbol{\Lambda}) = \pm 1$, c'est-à-dire $\boldsymbol{\Lambda}$ est unimodulaire.

$3 \Rightarrow 1$: Comme $\boldsymbol{U}$ est unimodulaire, $\boldsymbol{U}^{-1}$ l'est aussi. Or, $\boldsymbol{A} = \boldsymbol{U}^{-1}\boldsymbol{B}$. $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ s'écrivent donc comme des combinaisons linéaires entières de $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, qui forment donc une base de $\mathcal{L}$.

□

Le théorème 27.3 signifie que le déterminant d'une base est un invariant du module, au signe près. Nous appellerons $|\det(\boldsymbol{A})|$ le *déterminant du module* $\mathcal{L}$, que l'on notera $\det \mathcal{L}$. Observons que $\det \mathcal{L}$ est le volume du parallélépipède défini par les vecteurs de la base. Le théorème 27.3 nous dit aussi que l'on peut passer d'une base à une autre par des transformations unimodulaires. Nous l'utiliserons dans nos algorithmes.

La base idéale pour notre problème est orthogonale, car toute base orthogonale contient un plus court vecteur de $\mathcal{L}$ (voir exercice 27.1). Cependant, une telle base n'existe pas toujours. Par exemple, le module bidimensionnel suivant n'admet pas de base orthogonale (les deux parallélogrammes en grisé ont pour volume $\det \mathcal{L}$).



Nous noterons $\|\boldsymbol{a}\|$ la norme euclidienne d'un vecteur $\boldsymbol{a}$. Rappelons que l'inégalité de Hadamard affirme que, pour toute matrice $n \times n$ réelle $\boldsymbol{A}$,

$$|\det(\boldsymbol{A})| \leqslant \|\boldsymbol{a}_1\| \cdots \|\boldsymbol{a}_n\|.$$

De plus, cette inégalité est une égalité ssi une ligne de $\boldsymbol{A}$ est nulle ou bien toutes les lignes sont deux à deux orthogonales. Ainsi, pour toute base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ d'un module $\mathcal{L}$,

$$\det \mathcal{L} \leqslant \|\boldsymbol{b}_1\| \cdots \|\boldsymbol{b}_n\|.$$

Puisqu'aucun des vecteurs n'est nul, cette inégalité est une égalité ssi la base est orthogonale. Nous appellerons *défaut d'orthogonalité* d'une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ la quantité :

$$\frac{\|\boldsymbol{b}_1\| \cdots \|\boldsymbol{b}_n\|}{\det \mathcal{L}}.$$

Puisque $\det \mathcal{L}$ est un invariant, plus le défaut d'orthogonalité est faible, plus les vecteurs doivent être courts.

Nous dirons qu'une famille de vecteurs $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathcal{L}$ linéairement indépendants est *primitive* si on peut la compléter en une base de $\mathcal{L}$. Nous dirons qu'un vecteur $\boldsymbol{a} \in \mathcal{L}$ est *le plus court dans sa direction* si $x\boldsymbol{a}$ n'est pas un vecteur de $\mathcal{L}$ pour tout $0 < |x| < 1$. Il est facile de déterminer si un vecteur seul est primitif.

Théorème 27.4 *Un vecteur $\boldsymbol{a} \in \mathcal{L}$ est primitif ssi $\boldsymbol{a}$ est le plus court dans sa direction.*

Preuve : Supposons qu'il existe une base $\boldsymbol{B}$ de $\mathcal{L}$ contenant $\boldsymbol{a}$. Tout vecteur de $\mathcal{L}$ de la forme $x\boldsymbol{a}$, où x est un scalaire, est nécessairement engendré par $\boldsymbol{B}$, et donc $x \in \mathbf{Z}$. Ainsi, $\boldsymbol{a}$ est le plus court dans sa direction.

Supposons que $\boldsymbol{a}$ est le plus court dans sa direction. Puisque $\boldsymbol{a} \in \mathcal{L}$, nous pouvons écrire :

$$\boldsymbol{a} = \lambda_1 \boldsymbol{a}_1 + \cdots + \lambda_n \boldsymbol{a}_n,$$

où $\lambda_i \in \mathbf{Z}$ et $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ est une base de $\mathcal{L}$. Puisque $\boldsymbol{a}$ est le plus court dans sa direction, $\operatorname{pgcd}(\lambda_1, \ldots, \lambda_n)$ vaut 1. Par conséquent, il existe une matrice $n \times n$ unimodulaire $\boldsymbol{\Lambda}$, dont la première ligne est $\lambda_1, \ldots, \lambda_n$ (voir exercice 27.2). Posons $\boldsymbol{B} = \boldsymbol{\Lambda}\boldsymbol{A}$. D'après le théorème 27.3, $\boldsymbol{B}$ est une base de $\mathcal{L}$, qui contient $\boldsymbol{a}$, CQFD. □

27.2 Les algorithmes d'Euclide et de Gauss

En dimension 1, le module engendré par un unique vecteur $\boldsymbol{a}$ est l'ensemble des multiples de $\boldsymbol{a}$. Le problème du vecteur le plus court en dimension 1 est donc trivial. Considérons plutôt le problème suivant : étant donné deux entiers a et b, considérer l'ensemble des entiers obtenus par combinaison linéaire entière de a et b, et trouver le plus petit entier positif de ce module. Il s'agit du pgcd de a et b, noté $a \curlywedge b$.

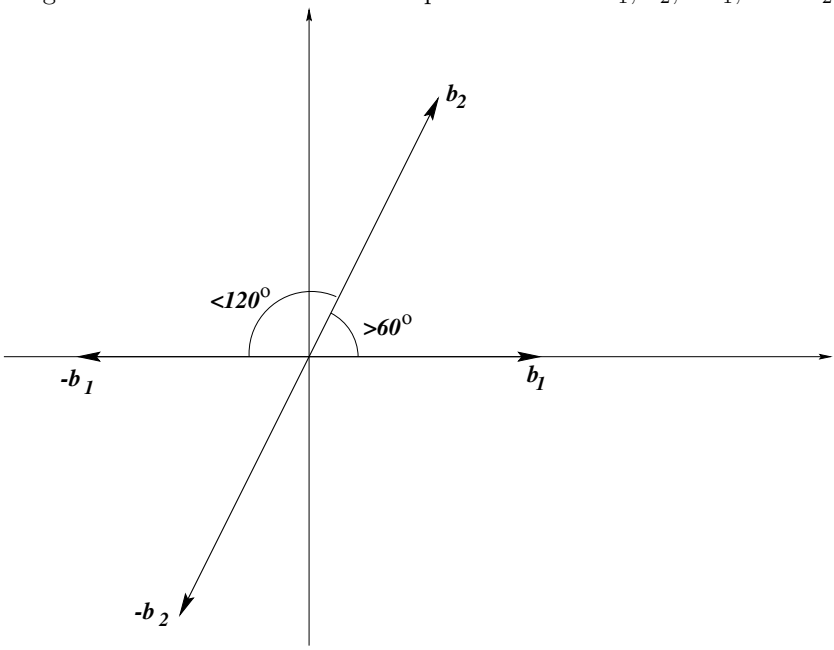
Quitte à échanger a et b, supposons que $a \geqslant b \geqslant 0$. L'idée de l'algorithme d'Euclide est de remplacer les entrées originales du problème par d'autres plus petites, en remarquant que $a \curlywedge b = (a - b) \curlywedge b$. En itérant, nous sommes ramenés à trouver le plus petit entier, en valeur absolue, de l'ensemble $\{|a - mb| : m \in \mathbf{Z}\}$. Notons-le c. Si $c = 0$, alors $a \curlywedge b = b$, et nous avons terminé. Sinon $a \curlywedge b = b \curlywedge c$, et nous itérons sur la paire (b, c). Comme $c \leqslant b/2$, ce processus termine au bout d'au plus $\log_2 b$ itérations.

Étudions maintenant l'algorithme de Gauss en dimension 2. En dimension 2, il existe une condition plus faible que l'orthogonalité pour garantir qu'une base contient un vecteur le plus court. Notons θ l'angle entre les deux vecteurs

de la base $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$, $0° < \theta < 180°$. Alors, $\det \mathcal{L} = \|\boldsymbol{b}_1\| \cdot \|\boldsymbol{b}_2\| \sin\theta$. Quitte à échanger b_1 et b_2, supposons $\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_2\|$.

Théorème 27.5 *Si* $60° \leqslant \theta \leqslant 120°$*, alors* $\boldsymbol{b}_1$ *est un vecteur le plus court du module* $\mathcal{L}$*.*

Preuve : Supposons par l'absurde qu'il existe un vecteur $\boldsymbol{b} \in \mathcal{L}$ plus court que $\boldsymbol{b}_1$. Puisque $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$ sont tous deux primitifs, $\boldsymbol{b}$ ne peut pas être multiple de $\boldsymbol{b}_1$ ou $\boldsymbol{b}_2$. Une simple énumération des cas, montre que tout vecteur $\boldsymbol{b}$ fait un angle inférieur à 60° avec l'un des quatre vecteurs $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $-\boldsymbol{b}_1$, ou $-\boldsymbol{b}_2$.



Notons $\boldsymbol{D}$ la matrice 2×2 dont les lignes sont $\boldsymbol{b}$ et le vecteur, parmi $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $-\boldsymbol{b}_1$, ou $-\boldsymbol{b}_2$, qui fait un angle $< 60°$ avec $\boldsymbol{b}$. Remarquons que $|\det(\boldsymbol{D})|$ est non nul et qu'il est strictement inférieur à $\det \mathcal{L} = \|\boldsymbol{b}_1\| \cdot \|\boldsymbol{b}_2\| \sin\theta$, ce qui contredit que $\det(\boldsymbol{D})$ est un multiple entier de $\det \mathcal{L}$. $\boldsymbol{b}_1$ est donc bien un vecteur le plus court de $\mathcal{L}$. □

Notons le produit scalaire par $\cdot$ et posons

$$\mu_{21} = \frac{\boldsymbol{b}_2 \cdot \boldsymbol{b}_1}{\|\boldsymbol{b}_1\|^2}.$$

Observons que $\mu_{21}\boldsymbol{b}_1$ est la projection de $\boldsymbol{b}_2$ sur la direction de $\boldsymbol{b}_1$. La proposition suivante suggère un algorithme pour trouver une base satisfaisant la condition du théorème 27.5.

Proposition 27.6 *Si une base* $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ *vérifie*
– $\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_2\|$ *et*

– $|\mu_{21}| \leqslant 1/2$,
alors $60^\circ \leqslant \theta \leqslant 120^\circ$.

Preuve : Remarquons que

$$\cos\theta = \frac{\boldsymbol{b}_1 \cdot \boldsymbol{b}_2}{\|\boldsymbol{b}_1\| \cdot \|\boldsymbol{b}_2\|} = \frac{\mu_{21}\|\boldsymbol{b}_1\|}{\|\boldsymbol{b}_2\|}.$$

Ainsi, d'après les deux conditions, $|\cos\theta| \leqslant 1/2$. D'où, $60^\circ \leqslant \theta \leqslant 120^\circ$. □

Gauss proposa l'algorithme suivant pour transformer une base arbitraire en une base qui satisfait les conditions énoncées ci-dessus.

Algorithme 27.7 (Vecteur le plus court en dimension 2)

1. Initialisation : quitte à échanger $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$, $\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_2\|$.
2. Tant que les conditions de la proposition 27.6 ne sont pas vraies, faire :
   (a) Si $|\mu_{21}| > 1/2$, faire $\boldsymbol{b}_2 \leftarrow \boldsymbol{b}_2 - m\boldsymbol{b}_1$, où m est l'entier le plus proche de μ_{21}.
   (b) Si $\|\boldsymbol{b}_1\| > \|\boldsymbol{b}_2\|$, échanger $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$.
3. Renvoyer $\boldsymbol{b}_1$.

Remarquons que les opérations en jeu consistent à échanger les lignes de $\boldsymbol{B}$ et à soustraire un multiple d'une ligne à l'autre. Ces opérations sont clairement unimodulaires (la valeur absolue du déterminant est inchangée). Par conséquent, nous conservons bien une base de $\mathcal{L}$.

Clairement, après l'étape 2(a), $|\mu_{21}| \leqslant 1/2$. Remarquons que cette étape est très similaire à l'algorithme du pgcd d'Euclide. Elle minimise la projection de $\boldsymbol{b}_2$ sur $\boldsymbol{b}_1$ en soustrayant à $\boldsymbol{b}_2$ un nombre convenable de fois $\boldsymbol{b}_1$. Puisque $\|\boldsymbol{b}_1\| \cdot \|\boldsymbol{b}_2\|$ décroît à chaque itération, l'algorithme doit terminer (il n'y a qu'un nombre fini de vecteurs de $\mathcal{L}$ à l'intérieur d'une boule donnée). La preuve que l'algorithme 27.7 termine en temps polynomial est donnée dans les exercices 27.3, 27.4, et 27.5, et dans les commentaires de la section 27.7.

27.3 Minorer OPT par l'orthogonalisation de Gram-Schmidt

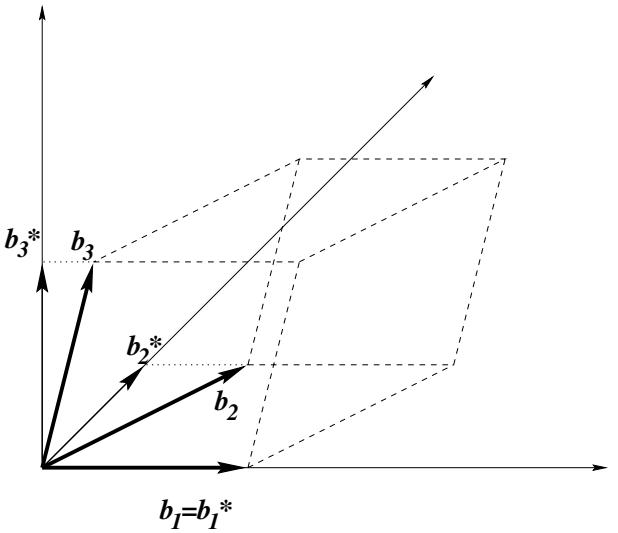
Nous présentons dans cette section le minorant de Gram-Schmidt pour OPT, c'est-à-dire pour la longueur du plus court vecteur du module $\mathcal{L}$.

Intuitivement, l'*orthogonalisation de Gram-Schmidt* de la base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ donne les n « hauteurs » du parallélépipède défini par cette base. Formellement, il s'agit d'un ensemble de vecteurs 2 à 2 orthogonaux $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$,

tels que $\boldsymbol{b}_1^* = \boldsymbol{b}_1$ et $\boldsymbol{b}_i^*$ est la composante de $\boldsymbol{b}_i$ orthogonale à $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_{i-1}^*$, pour $2 \leqslant i \leqslant n$. $\boldsymbol{b}_i^*$ s'obtient en soustrayant à $\boldsymbol{b}_i$ ses composantes selon les directions de $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_{i-1}^*$; nous avons la relation de récurrence suivante :

$$\boldsymbol{b}_1^* = \boldsymbol{b}_1$$

$$\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \frac{\boldsymbol{b}_i \cdot \boldsymbol{b}_j^*}{\|\boldsymbol{b}_j^*\|^2} \boldsymbol{b}_j^*, \quad i = 2, \ldots, n. \tag{27.1}$$



Pour $1 \leqslant j < i \leqslant n$, posons :

$$\mu_{ij} = \frac{\boldsymbol{b}_i \cdot \boldsymbol{b}_j^*}{\|\boldsymbol{b}_j^*\|^2},$$

et $\mu_{ii} = 1$. Alors,

$$\boldsymbol{b}_i = \sum_{j=1}^{i} \mu_{ij} \boldsymbol{b}_j^*, \quad i = 1, \ldots, n. \tag{27.2}$$

Pour $j \leqslant i$, nous noterons $\boldsymbol{b}_i(j)$ la composante de $\boldsymbol{b}_i$ orthogonale à $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{j-1}$, c'est-à-dire

$$\boldsymbol{b}_i(j) = \mu_{ij} \boldsymbol{b}_j^* + \mu_{i,j+1} \boldsymbol{b}_{j+1}^* + \cdots + \boldsymbol{b}_i^*.$$

Il est facile de montrer que :

$$\det \mathcal{L} = \|\boldsymbol{b}_1^*\| \cdots \|\boldsymbol{b}_n^*\|.$$

Le défaut d'orthogonalité de la base peut donc se réécrire :

$$\frac{\|\boldsymbol{b}_1\| \cdots \|\boldsymbol{b}_n\|}{\|\boldsymbol{b}_1^*\| \cdots \|\boldsymbol{b}_n^*\|} = \frac{1}{\sin\theta_2 \cdots \sin\theta_n},$$

où θ_i est l'angle que forme $\boldsymbol{b}_i$ avec le sous-espace vectoriel engendré par $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$, pour $2 \leqslant i \leqslant n$. Cet angle est défini ainsi : soit $\boldsymbol{b}_i'$ la projection de $\boldsymbol{b}_i$ sur le sous-espace engendré par $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$, alors

$$\theta_i = \arccos\left(\frac{\|\boldsymbol{b}_i'\|}{\|\boldsymbol{b}_i\|}\right).$$

Remarquons que l'orthogonalisation de Gram-Schmidt dépend non seulement de la base initiale, mais aussi de l'ordre des vecteurs dans cette base. Ce procédé nous fournit la minoration suivante de OPT.

Lemme 27.8 *Soit $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ une base du module $\mathcal{L}$, et $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$ la base orthogonale de Gram-Schmidt associée. Alors,*

$$\mathrm{OPT} \geqslant \min\{\|\boldsymbol{b}_1^*\|, \ldots, \|\boldsymbol{b}_n^*\|\}.$$

Preuve : Soit $\boldsymbol{v}$ un vecteur le plus court de $\mathcal{L}$. Soit k l'indice du dernier vecteur de la base utilisé pour écrire $\boldsymbol{v}$, c'est-à-dire tel que :

$$\boldsymbol{v} = \sum_{i=1}^{k} \lambda_i \boldsymbol{b}_i,$$

avec $\lambda_k \neq 0$. Avec (27.2), on peut réécrire $\boldsymbol{v}$ comme combinaison linéaire des vecteurs $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*$. Dans cette combinaison, le cœfficient de $\boldsymbol{b}_k^*$ est λ_k, puisque $\mu_{k,k} = 1$. Or les vecteurs $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$ sont orthogonaux, donc

$$\|\boldsymbol{v}\|^2 \geqslant \lambda_k^2 \|\boldsymbol{b}_k^*\|^2 \geqslant \|\boldsymbol{b}_k^*\|^2.$$

Enfin,

$$\mathrm{OPT} = \|\boldsymbol{v}\| \geqslant \|\boldsymbol{b}_k^*\| \geqslant \min\{\|\boldsymbol{b}_1^*\|, \ldots, \|\boldsymbol{b}_n^*\|\}. \qquad \square$$

27.4 Algorithme en dimension n

Cette section étend l'algorithme de Gauss au cas d'un module de dimension n. Comme d'habitude, nos efforts vont consister essentiellement à améliorer notre minorant de OPT. Rappelons que pour toute base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$,

$$\|\boldsymbol{b}_1^*\| \cdots \|\boldsymbol{b}_n^*\| = \det \mathcal{L}$$

est un invariant du module, et que $\min\{\|\boldsymbol{b}_1^*\|, \ldots, \|\boldsymbol{b}_n^*\|\}$ minore OPT. Typiquement, le minimum d'une base est atteint vers la fin de la séquence $\|\boldsymbol{b}_1^*\|, \ldots, \|\boldsymbol{b}_n^*\|$. Nous allons essayer d'assurer que cette séquence reste « lexicographiquement petite », et de garantir ainsi que les dernières entrées, et donc le minimum, ne soit pas trop petit. Les conditions de la proposition 27.6 suggèrent une façon de procéder. En dimension 2, la première condition nous dit que :

$$\|\boldsymbol{b}_1\|^2 \leqslant \|\boldsymbol{b}_2\|^2, \quad \text{c'est-à-dire} \quad \|\boldsymbol{b}_1^*\|^2 \leqslant \mu_{21}^2 \|\boldsymbol{b}_1^*\|^2 + \|\boldsymbol{b}_2^*\|^2.$$

En utilisant la borne sur μ_{21} de la seconde condition, nous obtenons :

$$\|\boldsymbol{b}_1^*\|^2 \leqslant \frac{1}{4}\|\boldsymbol{b}_1^*\|^2 + \|\boldsymbol{b}_2^*\|^2.$$

Ainsi,

$$\|\boldsymbol{b}_1^*\| \leqslant \frac{2}{\sqrt{3}}\,\|\boldsymbol{b}_2^*\|.$$

$\boldsymbol{b}_2^*$ ne peut donc être plus de $2/\sqrt{3}$ fois plus petit que $\boldsymbol{b}_1^*$. Nous allons imposer des conditions similaires à chaque paire de vecteurs consécutifs de la base, de sorte que pour tout i, $\boldsymbol{b}_{i+1}^*$ ne soit pas plus de $2/\sqrt{3}$ fois plus court que $\boldsymbol{b}_i^*$. Ainsi, la longueur du premier vecteur, $\boldsymbol{b}_1$, sera bien à un facteur exponentiel du minorant. Itérer l'algorithme 27.7 permet d'assurer ces conditions. Cependant, nous ne savons pas prouver la terminaison de cette extension de l'algorithme (voir exercice 27.7 pour plus de détails).

Pour garantir la terminaison au bout d'un nombre polynomial d'itérations, nous utilisons une idée similaire à celle de l'exercice 27.3. Nous dirons qu'une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ est *Gauss-réduite* si, pour tout $1 \leqslant i \leqslant n-1$,

- $\|\boldsymbol{b}_i(i)\| \leqslant \frac{2}{\sqrt{3}}\,\|\boldsymbol{b}_{i+1}(i)\|$, et
- $|\mu_{i+1,i}| \leqslant 1/2$.

L'algorithme pour obtenir une base Gauss-réduite est une généralisation simple de l'algorithme 27.7.

Algorithme 27.9 (Algorithme du vecteur le plus court)

1. Tant que la base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ n'est pas Gauss-réduite, faire :
   (a) Pour i allant de 1 à $n-1$, assurer que $|\mu_{i+1,i}| \leqslant 1/2$.
   Si $|\mu_{i+1,i}| > 1/2$ alors $\boldsymbol{b}_{i+1} \leftarrow \boldsymbol{b}_{i+1} - m\boldsymbol{b}_i$, où m est l'entier le plus proche de $\mu_{i+1,i}$.
   (b) Sélectionner un index i tel que $\|\boldsymbol{b}_i\| > \frac{2}{\sqrt{3}} \|\boldsymbol{b}_{i+1}(i)\|$, puis échanger $\boldsymbol{b}_i$ et $\boldsymbol{b}_{i+1}$.
2. Renvoyer $\boldsymbol{b}_1$.

Théorème 27.10 *L'algorithme 27.9 termine au bout d'un nombre polynomial d'itérations et réalise une $2^{(n-1)/2}$-approximation.*

Preuve : Pour $1 \leqslant i \leqslant n-1$, nous avons :

$$\begin{aligned}\|\boldsymbol{b}_i(i)\|^2 = \|\boldsymbol{b}_i^*\|^2 &\leqslant \frac{4}{3}\|\boldsymbol{b}_{i+1}(i)\|^2 = \frac{4}{3}(\mu_{i+1,i}^2\|\boldsymbol{b}_i^*\|^2 + \|\boldsymbol{b}_{i+1}^*\|^2) \\ &\leqslant \frac{4}{3}\left(\frac{1}{4}\|\boldsymbol{b}_i^*\|^2 + \|\boldsymbol{b}_{i+1}^*\|^2\right).\end{aligned}$$

Ainsi,

$$\|\boldsymbol{b}_i^*\|^2 \leqslant 2\|\boldsymbol{b}_{i+1}^*\|^2. \tag{27.3}$$

Et donc,

$$\|\boldsymbol{b}_1\| \leqslant 2^{(n-1)/2} \min_i \|\boldsymbol{b}_i^*\| \leqslant 2^{(n-1)/2}\ \mathrm{OPT}\,.$$

Pour démontrer la terminaison en temps polynomial de l'algorithme, nous utilisons la fonction potentiel suivante :

$$\Phi = \prod_{i=1}^{n} \|\boldsymbol{b}_i^*\|^{(n-i)}.$$

Remarquons que l'étape 1(a) de l'algorithme 27.9 ne change pas l'orthogonalisée de Gram–Schmidt et dure le temps de $O(n)$ opérations arithmétiques (addition, soustraction, multiplication ou division). Au plus un échange de vecteur a lieu à chaque itération (étape 1(b)). Lorsqu'un échange a lieu, Φ est divisée au moins par $2/\sqrt{3}$. C'est pour cette raison que nous avons introduit le facteur $2/\sqrt{3}$ dans la définition d'une base Gauss-réduite. Supposons, sans perte de généralité, que les composantes de la base initiale sont entières.

Alors, la valeur initiale de Φ est inférieure à :

$$(\max_i \|\boldsymbol{b}_i\|)^{n(n-1)/2}.$$

Démontrons que $\Phi \geqslant 1$ tout au long de l'exécution de l'algorithme. Ainsi, si le nombre d'itérations est m, alors,

$$\left(\frac{2}{\sqrt{3}}\right)^m \leqslant (\max_i \|\boldsymbol{b}_i\|)^{n(n-1)/2}.$$

Et donc,

$$m \leqslant \frac{n(n-1)\max_i \log \|\boldsymbol{b}_i\|}{2\log(2/\sqrt{3})}.$$

Écrivons les vecteurs de la base $\boldsymbol{b}_1, \ldots \boldsymbol{b}_n$ dans la base orthogonale $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$. Pour $1 \leqslant k \leqslant n-1$, notons $\boldsymbol{B}_k$ la matrice triangulaire inférieure $k \times k$ dont les lignes sont les $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$. Clairement,

$$\det(\boldsymbol{B}_k) = \|\boldsymbol{b}_1^*\| \cdots \|\boldsymbol{b}_k^*\|.$$

Or, le cœfficient (i,j) de $\boldsymbol{B}_k \boldsymbol{B}_k^T$ vaut $\boldsymbol{b}_i \cdot \boldsymbol{b}_j$, qui est entier. Ainsi,

$$\det(\boldsymbol{B}_k \boldsymbol{B}_k^T) = \|\boldsymbol{b}_1^*\|^2 \cdots \|\boldsymbol{b}_k^*\|^2 \geqslant 1.$$

Finalement,

$$\Phi = \prod_{k=1}^{(n-1)} \det(\boldsymbol{B}_k \boldsymbol{B}_k^T) \geqslant 1.$$

□

Nous avons démontré que l'algorithme 27.9 termine après un nombre polynomial d'opérations arithmétiques. Pour être rigoureux, nous devons borner le nombre d'opérations sur les bits, c'est-à-dire nous devons montrer que les nombres impliqués s'écrivent avec un nombre polynomial de bits. Nous définissons ci-dessous la notion plus forte de base Lovász-réduite. Nous pourrions démontrer qu'une modification simple de l'algorithme 27.9 produit une base Lovász-réduite au bout d'un nombre polynomial d'opérations sur les bits. Cependant, nous préférons omettre cette preuve qui est pénible.

L'intérêt des bases Lovász-réduite est qu'on sait borner leurs défauts d'orthogonalité. Nous dirons qu'une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ est *réduite au sens faible* si $|\mu_{ij}| \leqslant 1/2$, pour tous $1 \leqslant j \leqslant i \leqslant n$. Nous dirons qu'une base est *Lovász-réduite* si elle est Gauss-réduite et réduite au sens faible. On peut obtenir une base réduite au sens faible à partir de l'orthogonalisation de Gram-Schmidt, avec moins de $n(n-1)/2$ opérations (voir exercice 27.9). En remplaçant

l'étape 1(a) de l'algorithme 27.9 par cette procédure, nous obtenons un algorithme qui construit une base Lovász-réduite. Les bornes sur le nombre d'itérations obtenues au théorème 27.10 s'appliquent aussi à ce nouvel algorithme.

Théorème 27.11 *Le défaut d'orthogonalité d'une base Lovász-réduite est inférieur à* $2^{n(n-1)/4}$.

Preuve : Nous utilisons l'inégalité (27.3) établie au théorème 27.10.

$$\|\boldsymbol{b}_i\|^2 = \sum_{j=1}^{i} \mu_{ij}^2 \|\boldsymbol{b}_j^*\|^2 \leqslant \sum_{j=1}^{i-1} \frac{1}{4}\|\boldsymbol{b}_j^*\|^2 + \|\boldsymbol{b}_i^*\|^2$$

$$\leqslant (1 + \frac{1}{4}(2 + \cdots + 2^{i-1}))\|\boldsymbol{b}_i^*\|^2 \leqslant 2^{i-1}\|\boldsymbol{b}_i^*\|^2.$$

Ainsi,

$$\prod_{i=1}^{n} \|\boldsymbol{b}_i\|^2 \leqslant 2^{n(n-1)/2} \prod_{i=1}^{n} \|\boldsymbol{b}_i^*\|^2 = 2^{n(n-1)/2}(\det \mathcal{L})^2.$$

CQFD. □

Exemple 27.12 Voici une instance critique. Considérons la base suivante ainsi que son orthogonalisée de Gram-Schmidt.

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 1/2 & \rho & 0 & \dots & 0 & 0 \\ 1/2 & \rho/2 & \rho^2 & \dots & 0 & 0 \\ \vdots & & & & & \\ 1/2 & \rho/2 & \rho^2/2 & \dots & \rho^{n-2}/2 & \rho^{n-1} \end{pmatrix}$$

Posons $\rho = \sqrt{3}/2$. Il est facile de vérifier que c'est une base Lovász-réduite. Les vecteurs de la base sont tous unitaires. Le minorant de Gram-Schmidt est $\rho^{(n-1)}$, c'est-à-dire exponentiellement plus petit que la longueur de n'importe quel vecteur de la base. En soustrayant la dernière ligne à l'avant-dernière, nous obtenons le vecteur :

$$\left(0, \dots, 0, \frac{\rho^{(n-2)}}{2}, -\rho^{(n-1)}\right),$$

qui est exponentiellement plus petit que n'importe quel vecteur de la base. □

27.5 Le module dual et ses applications algorithmiques

À l'instar de la programmation linéaire, on peut associer à tout module une structure qui peut être interprétée comme son dual. La puissance de la dualité en programmation linéaire repose sur l'existence d'une relation min-max entre un programme et son dual. Il n'existe pas de telle relation entre un module et son module dual. En fait, la dualité ne semble pas être d'une grande utilité algorithmiquement parlant pour les modules. Dans cette section, nous allons néanmoins utiliser la dualité pour démontrer que le minorant de Gram-Schmidt est un bon minorant de la valeur optimum en produisant pour *toute* instance, une base pour laquelle ce minorant est supérieur à OPT/n.

Le *module dual* $\mathcal{L}^*$ de $\mathcal{L}$ est l'ensemble $\mathcal{L}^* = \{\boldsymbol{v} \in \mathbf{R}^n : \forall \boldsymbol{b} \in \mathcal{L},\ \boldsymbol{b}\cdot\boldsymbol{v} \in \mathbf{Z}\}$. Pour toute matrice $n \times n$ $\boldsymbol{B}$, nous noterons $\boldsymbol{B}^{-T}$ la matrice $(\boldsymbol{B}^{-1})^T$.

Théorème 27.13 *Pour toute base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ de $\mathcal{L}$, les lignes de la matrice $\boldsymbol{B}^{-T}$ forment une base du module dual $\mathcal{L}^*$. De plus,* $\det \mathcal{L}^* = 1/\det \mathcal{L}$.

Preuve : Soient $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ les lignes de $\boldsymbol{B}^{-T}$. Par construction,

$$\boldsymbol{b}_i \cdot \boldsymbol{v}_j = \begin{cases} 1 \text{ si } i = j \\ 0 \text{ sinon.} \end{cases}$$

Ainsi, le produit scalaire de toute combinaison linéaire entière des $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ avec chacun des $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ est entier. Le module engendré par les $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ est donc inclus dans $\mathcal{L}^*$. Réciproquement, soit $\boldsymbol{v} \in \mathcal{L}^*$. Posons $a_i = \boldsymbol{v}\cdot\boldsymbol{b}_i \in \mathbf{Z}$, pour $1 \leqslant i \leqslant n$, et $\boldsymbol{a} = (a_1, \ldots, a_n)$. Par construction, $\boldsymbol{B}\boldsymbol{v}^T = \boldsymbol{a}^T$, et donc $\boldsymbol{v} = \boldsymbol{a}\boldsymbol{B}^{-T}$, c'est-à-dire $\boldsymbol{v}$ s'écrit comme une combinaison linéaire entière des vecteurs $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$, qui forment donc bien une base de $\mathcal{L}^*$. Enfin,

$$\det \mathcal{L}^* = \det(\boldsymbol{B}^{-T}) = \frac{1}{\det \mathcal{L}}.$$ □

Soit $\boldsymbol{v} \in \mathbf{R}^n$ un vecteur non nul. Notons $\boldsymbol{v}^\perp$ le sous-espace vectoriel $\{\boldsymbol{b} \in \mathbf{R}^n : \boldsymbol{b}\cdot\boldsymbol{v} = 0\}$ orthogonal à v, de dimension $n-1$. Nous appellerons *sous-module* de $\mathcal{L}$, tout module $\mathcal{L}' \subset \mathcal{L}$. La *dimension* de $\mathcal{L}'$ est définie comme celle du sous-espace vectoriel engendré par $\mathcal{L}'$. Le lemme 27.14 et l'exercice 27.11 démontrent qu'il existe une bijection entre les sous-modules de $\mathcal{L}$ de dimension $n-1$ et les vecteurs primitifs de $\mathcal{L}^*$.

Lemme 27.14 *Pour tout vecteur primitif $\boldsymbol{v} \in \mathcal{L}^*$,*
- *$\mathcal{L} \cap (\boldsymbol{v}^\perp)$ est un sous-module de $\mathcal{L}$ de dimension $(n-1)$, et*
- *il existe un vecteur $\boldsymbol{b} \in \mathcal{L}$ tel que $\boldsymbol{v}\cdot\boldsymbol{b} = 1$.*

Preuve : Comme $\boldsymbol{v}$ est primitif, il existe une base $\boldsymbol{v}, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ de $\mathcal{L}^*$. Notons $\boldsymbol{V}$ la matrice $n \times n$ dont les lignes sont $\boldsymbol{v}, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$. Soient $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ les lignes de $\boldsymbol{V}^{-T}$. Comme $\boldsymbol{v}\cdot\boldsymbol{b}_i = 0$, pour tout $2 \leqslant i \leqslant n$, le sous-module

$(n-1)$-dimensionnel de $\mathcal{L}$ engendré par les $\boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$ est inclus dans $\boldsymbol{v}^\perp$. Enfin, il suffit de remarquer que $\boldsymbol{v} \cdot \boldsymbol{b}_1 = 1$ pour la seconde partie de la proposition. □

Voici une procédure clé pour construire une base à partir d'une bonne orthogonalisation de Gram-Schmidt. Prenons $\boldsymbol{v} \in \mathcal{L}^*$ un vecteur primitif, $\mathcal{L}' = \mathcal{L} \cap (\boldsymbol{v}^\perp)$, et $\boldsymbol{w} \in \mathcal{L}$ tel que $\boldsymbol{v} \cdot \boldsymbol{w} = 1$. Soient $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$ une base de $\mathcal{L}'$ et $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_{n-1}^*$ sa base orthogonale de Gram-Schmidt associée.

Lemme 27.15 *$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{w}$ est une base de $\mathcal{L}$, dont l'orthogonalisée de Gram-Schmidt est :*

$$\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_{n-1}^*, \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|^2}.$$

Preuve : Soit $\boldsymbol{b} \in \mathcal{L}$ un vecteur. Remarquons que $\boldsymbol{b} - (\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{w} \in \mathcal{L}'$, car $(\boldsymbol{b} - (\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{w}) \cdot \boldsymbol{v} = 0$. Ainsi, nous pouvons réécrire :

$$\boldsymbol{b} - (\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{w} = \sum_{i=1}^{n-1} \lambda_i \boldsymbol{b}_i,$$

où $\lambda_1, \ldots, \lambda_{n-1} \in \mathbf{Z}$. D'où,

$$\boldsymbol{b} = \sum_{i=1}^{n-1} \lambda_i \boldsymbol{b}_i + (\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{w}.$$

Ainsi, chaque vecteur de $\mathcal{L}$ s'écrit comme une combinaison linéaire entière de $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{w}$, CQFD pour la première partie.

Notons $\mu_i \boldsymbol{b}_i^*$, la composante de $\boldsymbol{w}$ dans la direction de $\boldsymbol{b}_i^*$, pour $1 \leqslant i \leqslant n-1$, et :

$$\boldsymbol{w}^* = \boldsymbol{w} - \sum_{i=1}^{n-1} \mu_i \boldsymbol{b}_i^*.$$

Comme $\boldsymbol{w}^*$ est orthogonal à chacun des $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$, il est colinéaire à $\boldsymbol{v}$. Or $\boldsymbol{v} \cdot \boldsymbol{w} = 1$, donc $\boldsymbol{v} \cdot \boldsymbol{w}^* = 1$. Et donc,

$$\|\boldsymbol{w}^*\| = \frac{1}{\|\boldsymbol{v}\|}.$$

D'où,

$$\boldsymbol{w}^* = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|^2}.$$ □

Voici comment nous allons utiliser le vecteur primitif construit ci-dessus. Posons $\mathcal{L}_1 = \mathcal{L}$, et notons $\mathcal{L}_1^*$ son dual. Choisissons un vecteur primitif $\boldsymbol{v}_1 \in \mathcal{L}_1^*$, et considérons le sous-module $\mathcal{L}_2 = \mathcal{L}_1 \cap (\boldsymbol{v}_1^\perp)$, et $\mathcal{L}_2^*$ son dual ($\mathcal{L}_2^*$ est, par définition, inclus dans le sous-espace vectoriel engendré par $\mathcal{L}_2$). Maintenant, continuons en sélectionnant un autre vecteur primitif $\boldsymbol{v}_2 \in \mathcal{L}_2^*$, et posons $\mathcal{L}_3 = \mathcal{L}_2 \cap (\boldsymbol{v}_2^\perp)$, ainsi de suite. À l'étape i, nous choisissons un vecteur primitif $\boldsymbol{v}_i \in \mathcal{L}_i^*$, et considérons le sous-module $\mathcal{L}_{i+1} = \mathcal{L}_i \cap (\boldsymbol{v}_i^\perp)$. Remarquons que même si $\mathcal{L}_i$ est un sous-module de $\mathcal{L}$, $\mathcal{L}_i^*$ n'est pas nécessairement un sous-module de $\mathcal{L}^*$. Puis, nous utilisons la deuxième partie du lemme 27.14 pour trouver un vecteur $\boldsymbol{w}_i \in \mathcal{L}_i$ tel que $\boldsymbol{v}_i \cdot \boldsymbol{w}_i = 1$, pour $1 \leqslant i \leqslant n$. Voici la procédure complète (nous parlons ici de « procédure » et non d'algorithme car nous ne savons pas comment exécuter certaines de ses étapes en temps polynomial).

Procédure 27.16

1. Initialisation : $\mathcal{L}_1 \leftarrow \mathcal{L}$, et $\mathcal{L}_1^* \leftarrow$ dual de $\mathcal{L}$.
2. Pour $i = 1$ à n faire :
   Sélectionner un vecteur primitif $\boldsymbol{v}_i \in \mathcal{L}_i^*$.
   Trouver $\boldsymbol{w}_i \in \mathcal{L}_i$ tel que $\boldsymbol{v}_i \cdot \boldsymbol{w}_i = 1$.
   $\mathcal{L}_{i+1} \leftarrow \mathcal{L}_i \cap (\boldsymbol{v}_i^\perp)$.
   $\mathcal{L}_{i+1}^* \leftarrow$ dual de $\mathcal{L}_{i+1}$.
3. Renvoyer $\boldsymbol{w}_n, \boldsymbol{w}_{n-1}, \ldots, \boldsymbol{w}_1$ et

$$\left(\frac{\boldsymbol{v}_n}{\|\boldsymbol{v}_n\|^2}, \ldots, \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|^2} \right).$$

En appliquant le lemme 27.15 par récurrence, nous obtenons que :

Lemme 27.17 *La famille de vecteurs $\boldsymbol{w}_n, \boldsymbol{w}_{n-1}, \ldots, \boldsymbol{w}_1$ renvoyée par la procédure 27.16 forme une base de $\mathcal{L}$, dont l'orthogonalisée de Gram-Schmidt est :*

$$\left(\frac{\boldsymbol{v}_n}{\|\boldsymbol{v}_n\|^2}, \ldots, \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|^2} \right).$$

Nous allons utiliser le théorème fondamental suivant, dû à Minkowski.

Théorème 27.18 *Il existe un vecteur $\boldsymbol{b} \in \mathcal{L}$ tel que $\|\boldsymbol{b}\| \leqslant \sqrt{n} \sqrt[n]{\det \mathcal{L}}$.*

Preuve : Notons $\|\boldsymbol{a}\|_\infty$ la norme ℓ_∞ d'un vecteur $\boldsymbol{a}$. Nous savons que $\|\boldsymbol{a}\| \leqslant \sqrt{n}\|\boldsymbol{a}\|_\infty$. Il suffit donc de montrer qu'il existe un vecteur $\boldsymbol{b} \in \mathcal{L}$, tel que $\|\boldsymbol{b}\|_\infty \leqslant \sqrt[n]{\det \mathcal{L}}$. Quite à appliquer une homothétie de rapport $1/\sqrt[n]{\det \mathcal{L}}$, supposons que $\det \mathcal{L} = 1$. Nous définissons la *densité* d'un module $\mathcal{L}'$, comme

le nombre de points du module par unité de volume ; clairement, la densité d'un module $\mathcal{L}'$ vaut $1/\det \mathcal{L}'$. Comme $\det \mathcal{L} = 1$, la densité de $\mathcal{L}$ vaut 1.

Notons $\mathcal{C} = \{\boldsymbol{v} \in \mathbf{R}^n : \|\boldsymbol{v}\|_\infty \leqslant 1/2\}$ le cube unitaire centré sur l'origine. Plaçons un tel cube sur chaque point de $\mathcal{L}$, c'est-à-dire considérons l'ensemble des cubes $\mathcal{C} + \boldsymbol{b}$, pour tout $\boldsymbol{b} \in \mathcal{L}$. Deux de ces cubes doivent s'intersecter. Sinon, par l'absurde, chaque cube ne contiendrait qu'un seul point du module et comme le volume de ces cubes vaut 1, la densité serait strictement inférieure à 1, contradiction.

Soient $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$ les centres des deux cubes qui s'intersectent. Alors, $\|\boldsymbol{b}_1 - \boldsymbol{b}_2\|_\infty \leqslant 1$, et $\boldsymbol{b} = \boldsymbol{b}_1 - \boldsymbol{b}_2$ est un point du module, CQFD. □

Théorème 27.19 *Il existe une base de $\mathcal{L}$ telle que le minorant de Gram-Schmidt associé est supérieur à* OPT/n.

Preuve : D'après le théorème 27.18, il existe deux vecteurs $\boldsymbol{b} \in \mathcal{L}$ et $\boldsymbol{v} \in \mathcal{L}^*$ tels que $\|\boldsymbol{b}\|\|\boldsymbol{v}\| \leqslant n \sqrt[n]{\det \mathcal{L} \cdot \det \mathcal{L}^*} = n$. Modifions la procédure 27.16 comme suit : prenons $\boldsymbol{v}_i \in \mathcal{L}_i^*$ un vecteur primitif, et $\boldsymbol{b}_i \in \mathcal{L}_i$ tel que $\|\boldsymbol{v}_i\|\|\boldsymbol{b}_i\| \leqslant n + 1 - i \leqslant n$; la définition des vecteurs $\boldsymbol{w}_i$ est inchangée.

D'après le lemme 27.17, $\boldsymbol{w}_n, \ldots, \boldsymbol{w}_1$ forment une base de $\mathcal{L}$ dont l'orthogonalisée de Gram-Schmidt est :

$$\left(\frac{\boldsymbol{v}_n}{\|\boldsymbol{v}_n\|^2}, \ldots, \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|^2} \right).$$

Soit $\boldsymbol{b}$ le vecteur le plus court parmi $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$. Alors,

$$\|\boldsymbol{b}\| = \min\{\|\boldsymbol{b}_1\|, \ldots, \|\boldsymbol{b}_n\|\} \leqslant n \min \left\{ \frac{1}{\|\boldsymbol{v}_1\|}, \ldots, \frac{1}{\|\boldsymbol{v}_n\|} \right\}.$$

Ainsi, le minorant de Gram-Schmidt vérifie :

$$\min \left\{ \frac{1}{\|\boldsymbol{v}_1\|}, \ldots, \frac{1}{\|\boldsymbol{v}_n\|} \right\} \geqslant \frac{\|\boldsymbol{b}\|}{n} \geqslant \frac{\mathrm{OPT}}{n}.$$

□

Une conséquence du théorème 27.19, est qu'il existe un *certificat négatif approché de facteur n* pour le problème du vecteur le plus court (voir définition section 1.2). Ainsi, si la réponse à la question *« la longueur du vecteur le plus court du module $\mathcal{L}$ est-elle inférieure à α ? »* est *« non »*, et qu'en fait $\alpha < \mathrm{OPT}/n$, alors il existe un certificat de taille polynomiale qui nous permet de vérifier en temps polynomial que la réponse est bien *« non »*. Ce certificat est la base dont l'existence est démontrée au théorème 27.19, et la vérification consiste simplement à vérifier que α est inférieur au minorant de Gram-Schmidt associé à cette base.

27.6 Exercices

27.1 Démontrez que si un module $\mathcal{L}$ admet une base orthogonale, alors le plus court vecteur de cette base est un vecteur le plus court de $\mathcal{L}$.
Indication : Exprimez la longueur d'un vecteur le plus court $\boldsymbol{v} \in \mathcal{L}$ en fonction des vecteurs de la base.

27.2 Rappelons que si $a \perp b = 1$, alors l'algorithme d'Euclide construit deux entiers x, y tels que $ax + by = 1$. Généralisez ce résultat en démontrant que : si $\text{pgcd}(a_1, \ldots, a_n) = 1$, alors il existe une matrice $n \times n$ unimodulaire $\boldsymbol{U}$ dont la première ligne est $a_1, \ldots, a_n$.
Indication : Procédez par récurrence sur n.

27.3 Pour démontrer que l'algorithme de Gauss termine en temps polynomial en dimension 2, relâchons la première condition en assurant seulement que le vecteur $\boldsymbol{b}_1$ n'est qu'au plus à peine plus grand que $\boldsymbol{b}_2$, c'est-à-dire que $\|\boldsymbol{b}_1\| \leqslant (1+\varepsilon)\|\boldsymbol{b}_2\|$. La seconde condition reste inchangée. Démontrez que l'algorithme termine après un nombre polynomial (en la longueur des vecteurs initiaux et $1/\varepsilon$) d'itérations. Quelle garantie obtenez-vous sur la longueur de $\boldsymbol{b}_1$ à la fin de l'algorithme ?
Indication : Utilisez le minorant de Gram-Schmidt pour établir la garantie.

27.4 La terminaison en temps polynomial de l'algorithme 27.7 est difficile à établir car à chaque itération, le progrès peut être minimal. Donnez un exemple qui démontre cette difficulté.
Indication : Considérez l'instance $\boldsymbol{b}_1 = (1, 0)$ et $\boldsymbol{b}_2 = (\frac{1}{2} + \varepsilon, \frac{\sqrt{3}}{2})$.

27.5 (Kaib et Schnorr [160]) Pour contourner la difficulté mentionnée à l'exercice 27.4, considérons la variante suivante de l'algorithme 27.7. Nous dirons qu'une base $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ est *bien ordonnée* si

$$\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_1 - \boldsymbol{b}_2\| < \|\boldsymbol{b}_2\|.$$

1. Supposons que $\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_2\|$. Démontrez que l'une des trois bases $(\boldsymbol{b}_1, \boldsymbol{b}_2)$, $(\boldsymbol{b}_1, \boldsymbol{b}_1 - \boldsymbol{b}_2)$, ou $(\boldsymbol{b}_1 - \boldsymbol{b}_2, \boldsymbol{b}_1)$ est bien ordonnée. Modifiez l'étape d'initialisation pour démarrer avec une base bien ordonnée.
2. Voici la procédure principale :
   (a) Si $|\mu_{21}| > 1/2$, faire $\boldsymbol{b}_2 \leftarrow \boldsymbol{b}_2 - m\boldsymbol{b}_1$, où m est l'entier le plus proche de μ_{21}.
   (b) Si $\|\boldsymbol{b}_1 - \boldsymbol{b}_2\| > \|\boldsymbol{b}_1 + \boldsymbol{b}_2\|$, faire $\boldsymbol{b}_2 \leftarrow -\boldsymbol{b}_2$.
   (c) Si $\|\boldsymbol{b}_1\| \leqslant \|\boldsymbol{b}_2\|$, arrêter et renvoyer $(\boldsymbol{b}_1, \boldsymbol{b}_2)$.
   Sinon, échanger $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$ et recommencer en (a).

   Démontrez que la base obtenue après l'étape (b) est bien ordonnée.

3. Supposons que cette variante de l'algorithme, partant d'une base bien ordonnée $(\boldsymbol{b}_1, \boldsymbol{b}_2)$, exécute k itérations avant de s'arrêter sur une base réduite. Numérotons à rebours les vecteurs rencontrés $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k, \boldsymbol{a}_{k+1}$, de sorte que $\boldsymbol{b}_1 = \boldsymbol{a}_k$, $\boldsymbol{b}_2 = \boldsymbol{a}_{k+1}$, et que la base finale est $(\boldsymbol{a}_1, \boldsymbol{a}_2)$. Démontrez que pour $i > 1$, $\|\boldsymbol{a}_i\| \leqslant \frac{1}{2}\|\boldsymbol{a}_{i+1}\|$, et donc que l'algorithme termine bien au bout d'un temps polynomial.

27.6 Exhibez un module $\mathcal{L}$ tridimensionnel et deux vecteurs $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathcal{L}$ tels que $\boldsymbol{b}_1$ et $\boldsymbol{b}_2$ soient primitifs individuellement, mais pas la famille $\boldsymbol{b}_1, \boldsymbol{b}_2$.

27.7 Supposons que la base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ vérifie, pour $1 \leqslant i \leqslant n-1$:
- $\|\boldsymbol{b}_i(i)\| \leqslant \|\boldsymbol{b}_{i+1}(i)\|$, et
- $|\mu_{i+1,i}| \leqslant 1/2$.

Démontrez que, pour $1 \leqslant i \leqslant n-1$:

$$\|\boldsymbol{b}_1\| \leqslant \left(\frac{2}{\sqrt{3}}\right)^i \|\boldsymbol{b}^*_{i+1}\|.$$

Et donc,

$$\|\boldsymbol{b}_1\| \leqslant \left(\frac{2}{\sqrt{3}}\right)^{n-1} \min_i\{\|\boldsymbol{b}^*_i\|\} \leqslant \left(\frac{2}{\sqrt{3}}\right)^{n-1} \mathrm{OPT}\,.$$

Indication : Montrez par un calcul similaire au cas bidimensionnel que, pour $1 \leqslant i \leqslant n-1$,

$$\|\boldsymbol{b}^*_i\| \leqslant \frac{2}{\sqrt{3}}\|\boldsymbol{b}^*_{i+1}\|.$$

27.8 Modifiez la définition de la Gauss-réduite pour obtenir, pour toute constante $c > 2/\sqrt{3}$, une $c^{(n-1)/2}$-approximation pour le problème du vecteur le plus court.
Indication : Remplacez le facteur $2/\sqrt{3}$ dans la définition de base Gauss-réduite par $1+\varepsilon$, pour un $\varepsilon > 0$ convenable.

27.9 Démontrez qu'on peut transformer toute base en une base réduite au sens faible en moins de $n(n-1)/2$ opérations arithmétiques.
Indication : L'ordre d'exécution des opérations de l'étape 1(a) de l'algorithme 27.9 n'a pas d'importance. En revanche, cet ordre est important pour obtenir une base réduite au sens faible. Sélectionnez une paire (i, j) avec $|\mu_{ij}| > 1/2$ et telle que j soit le plus grand possible, et appliquez l'opération de l'étape 1(a) de l'algorithme 27.9.

27.10 Prouvez que le module dual de $\mathcal{L}^*$ est $\mathcal{L}$.

27.11 Démontrez la réciproque du lemme 27.14.
- Si $\mathcal{L}'$ est un sous-module de dimension $(n-1)$ de $\mathcal{L}$, alors il existe un vecteur $\boldsymbol{v} \in \mathcal{L}^*$ tel que $\mathcal{L}' = \mathcal{L} \cap (\boldsymbol{v}^\perp)$.
- Pour tout $\boldsymbol{v} \in \mathcal{L}^*$, s'il existe $\boldsymbol{b} \in \mathcal{L}$ tel que $\boldsymbol{v} \cdot \boldsymbol{b} = 1$, alors $\boldsymbol{v}$ est primitif.

27.12 Soit $\boldsymbol{v} \in \mathcal{L}^*$ un vecteur primitif et $\mathcal{L}'$ un sous-module de dimension $(n-1)$ de $\mathcal{L}$ tel que $\mathcal{L}' = \mathcal{L} \cap (\boldsymbol{v}^\perp)$. Démontrez que $\det \mathcal{L}' = \|\boldsymbol{v}\| \cdot \det \mathcal{L}$.

27.13 Cet exercice propose une réciproque partielle du lemme 27.15. Soient $\boldsymbol{v}$ et $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$ définis comme au lemme 27.15. Démontrez que si $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{w}$ est une base de $\mathcal{L}$, alors $\boldsymbol{v} \cdot \boldsymbol{w} = 1$.
Indication : D'après le lemme 27.14, il existe un vecteur $\boldsymbol{w}' \in \mathcal{L}$ tel que $\boldsymbol{v} \cdot \boldsymbol{w}' = 1$. Exprimez $\boldsymbol{w}'$ sur la base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{w}$, et déduisez-en que $\boldsymbol{v} \cdot \boldsymbol{w} = 1$.

27.14 Démontrez que la base dont l'existence est établie au théorème 27.19 peut être trouvée en temps polynomial, à l'aide d'un oracle pour le problème du vecteur le plus court.
Indication : Un vecteur le plus court satisfait les conditions du théorème 27.18.

27.15 (Korkhine et Zolotarav [183]) Une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ d'un module $\mathcal{L}$ est dite *KZ-réduite* si :
- $\boldsymbol{b}_1$ est un vecteur le plus court de $\mathcal{L}$, $\boldsymbol{b}_2$ est tel que $\boldsymbol{b}_2^*$ est le plus court possible tel que la famille $\boldsymbol{b}_1, \boldsymbol{b}_2$ soit primitive, …, $\boldsymbol{b}_i$ est tel que $\boldsymbol{b}_i^*$ est le plus court possible tel que la famille $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i$ soit primitive, c'est-à-dire la composante de $\boldsymbol{b}_i$ orthogonale au sous-espace vectoriel engendré par $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$ est minimum, tout en assurant le caractère primitif de $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i$,
- et pour tout $1 \leqslant j < i \leqslant n$, $|\mu_{ij}| \leqslant 1/2$.

Proposez un algorithme polynomial qui trouve une base KZ-réduite d'un module, en utilisant un oracle pour le problème du vecteur le plus court.

27.16 (Lovász [201]) Soit $p(n)$ un polynôme. Démontrez que s'il existe un algorithme polynomial qui trouve un vecteur $\boldsymbol{v} \in \mathcal{L}$ tel que

$$\|\boldsymbol{v}\| \leqslant p(n) \sqrt[n]{\det \mathcal{L}},$$

alors il existe une $p^2(n)$-approximation pour le problème du vecteur le plus court.
Indication : L'idée principale est présentée dans le théorème 27.19.

27.17 (Conçu par K. Jain et R. Venkatesan, à partir de Frieze [95]) Étant donné un ensemble de n entiers positifs $S = \{a_1, \ldots, a_n\}$ et un entier b, le

problème de la somme partielle[4] consiste à trouver un sous-ensemble de S dont la somme est b. Ce problème **NP**-difficile permet de construire des fonctions à sens unique,[5] utilisées classiquement en cryptographie. Un message de n bits $c = c_1 \dots c_n$ est encodé par :

$$f_S(c) = \sum_{i=1}^{n} a_i c_i = b.$$

Clairement, f_S est facile à calculer. La **NP**-difficulté du problème de la somme partielle implique que f_S est difficile à inverser en général, c'est-à-dire étant donné $a_1, \dots, a_n$ et b, il est difficile de calculer c.

Cet exercice démontre comment, en utilisant l'algorithme 27.9, on peut inverser f_S avec forte probabilité, lorsque les $a_1, \dots, a_n$ sont tirés uniformément et indépendamment dans un (grand) intervalle $[1, B]$, avec $B \geqslant 2^{n^2}$. En pratique, il est utile de choisir B grand, pour que f_S soit injective.

Posons $m = 2^{n/2}\sqrt{n+1}$ et $p = m + 1$. Considérons la base suivante définissant un module $\mathcal{L}$ de dimension $n + 1$: $\boldsymbol{b}_0 = (p\,b, 0, ...0), \boldsymbol{b}_1 = (-p\,a_1, 1, 0...0), \dots, \boldsymbol{b}_n = (-p\,a_n, 0, 0, ..., 1)$.

1. Soient $u_1, \dots, u_n$ des entiers fixés. Démontrez que si les (a_i) sont tirés uniformément et indépendamment dans l'intervalle $[1, B]$, alors :

$$\mathbf{Pr}\left[\sum_{i=1}^{n} u_i a_i = 0\right] \leqslant \frac{1}{B},$$

où la probabilité est prise sur les choix possibles des (a_i).

2. Remarquons que $\mathcal{L}$ admet un vecteur non nul de longueur $\leqslant \sqrt{n}$. Démontrez que s'il n'existe pas d'autre vecteur non nul de longueur $\leqslant m$ dans $\mathcal{L}$, alors l'algorithme 27.9 permet de trouver c tel que $f_S(c) = b$.

3. Supposons qu'il existe un autre vecteur non nul $\boldsymbol{v} \in \mathcal{L}$ de longueur $\leqslant m$. Posons $\boldsymbol{v} = \lambda \boldsymbol{b}_0 + \sum_{i=1}^{n} v_i \boldsymbol{b}_i$, avec $\lambda, v_i \in \mathbf{Z}$. Clairement, $v_i \in [-m, m]$ pour tout i. Démontrez que $\lambda \in [-2m, 2m]$.

   Indication : Quitte à remplacer b par $\sum_{i=1}^{n} a_i - b$, supposons que $b \geqslant (\frac{1}{2}) \sum_{i=1}^{n} a_i$. La première coordonnée de $\boldsymbol{v}$ est nécessairement nulle, d'où :

$$|\lambda| b \leqslant \sum_{i=1}^{n} a_i |v_i| \leqslant (\sum_{i=1}^{n} a_i) \|\boldsymbol{v}\| \leqslant 2bm.$$

4. Démontrez que s'il existe un autre vecteur non nul dans $\mathcal{L}$ de longueur $\leqslant m$, alors il existe des entiers $u_1, \dots, u_n$ dans $[-3m, 3m]$, tels que $\sum_{i=1}^{n} u_i a_i = 0$.

[4] *Subset sum*, en anglais.
[5] *One-way function*, en anglais.

Indication : $\sum_{i=1}^{n} a_i(\lambda c_i - v_i) = 0$.

5. Démontrez que si les (a_i) sont tirés uniformément et indépendamment dans l'intervalle $[1, B]$, alors la probabilité que $\sum_{i=1}^{n} u_i a_i = 0$ pour des entiers $u_1, \ldots, u_n$ dans $[-3m, 3m]$, est très petite.
   Indication : Remarquez qu'il y a $(6m + 1)^n$ choix possibles pour $u_1, \ldots, u_n$.
6. Déduisez-en que f_S est inversée avec forte probabilité.

27.18 (Goldreich, Micciancio, Safra, et Seifert [115]) Étudions le *problème du vecteur le plus proche*.[6] Étant donné une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ d'un module $\mathcal{L}$ et un vecteur cible $\boldsymbol{b} \in \mathbf{R}^n$, trouver le vecteur du module qui soit le plus proche de $\boldsymbol{b}$ pour la distance euclidienne. Démontrez que pour toute fonction $f(n)$, une $f(n)$-approximation pour le problème du vecteur le plus proche permet de construire une $f(n)$-approximation pour le problème du vecteur le plus court.
Indication : Étant donné une base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ du module $\mathcal{L}$, supposons que $\boldsymbol{b} = \sum_i \lambda_i \boldsymbol{b}_i$ est un vecteur le plus court de $\mathcal{L}$. Clairement, les λ_i ne peuvent pas être tous pairs. Supposons que λ_i est impair. Considérons maintenant le module de base $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}, 2\boldsymbol{b}_i, \boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_n$. Observez que si le vecteur le plus proche de la cible $\boldsymbol{b}_i$ dans le module $\mathcal{L}'$ est $\boldsymbol{v}$, alors $\boldsymbol{b} = \boldsymbol{v} - \boldsymbol{b}_i$. Puisque i est inconnu, posez n fois cette question, et renvoyez la meilleure réponse.

27.7 Notes

L'algorithme 27.9 et le théorème 27.11 sont dus à Lenstra, Lenstra et Lovász [191]. Schnorr [244] modifia cet algorithme pour obtenir une $(1+\varepsilon)^n$-approximation, pour tout $\varepsilon > 0$. Plus précisément, l'algorithme de Schnorr est polynomial pour un choix convenable de $\varepsilon = o(1)$, ce qui donne un facteur d'approximation légèrement sous-exponentiel, c'est-à-dire une $2^{o(n)}$-approximation. C'est actuellement la meilleure garantie connue pour le problème du vecteur le plus court.

L'algorithme 27.7 a été publié par Gauss [106], qui l'énonça en termes de formes quadratiques. À l'aide d'une fonction potentiel complexe, Lagarias [188] montra que cet algorithme est polynomial (en le nombre de bits nécessaires codant la base en entrée). Reportez-vous à Kannan [162] pour un algorithme polynomial pour le problème du vecteur le plus court en dimension d, avec d constant. L'instance 27.12 est due à Kannan [161]. Ajtai [4] a montré que le problème du vecteur le plus court est **NP**-difficile par réduction randomisée. Le théorème 27.19 est dû à Lagarias, Lenstra et Schnorr [189].

[6] *Closest vector problem*, en anglais.

28 Problèmes de dénombrement

Les techniques de dénombrement approximatif des solutions de problèmes #**P**-complets sont assez différentes de celles développées dans les algorithmes d'approximation pour les problèmes d'optimisation **NP**-difficiles. Une grande partie de cette théorie est construite autour de la méthode de Monte-Carlo pour les chaînes de Markov (reportez-vous à la section 28.4 pour des références sur ce sujet). Dans ce chapitre, nous présentons deux algorithmes combinatoires (sans chaîne de Markov) pour deux problèmes fondamentaux : dénombrer les instanciations satisfaisant une formule sous forme normale disjonctive (DNF), et évaluer la probabilité de panne d'un réseau non orienté.

Intuitivement, la classe #**P** est l'ensemble des problèmes de dénombrement des solutions d'un problème **NP**. Formellement, soient L un langage de **NP**, M un vérificateur associé et p un polynôme majorant la taille de ses certificats positifs (voir section A.1). Nous associons à L une fonction f qui, à tout chaîne $x \in \Sigma^*$, associe le nombre de chaînes y telles que $|y| \leqslant p(|x|)$ et que $M(x, y)$ accepte. L'ensemble des fonctions $f : \Sigma^* \to \mathbf{N}$ ainsi construites forme la classe #**P**.

Une fonction $f \in \#\mathbf{P}$ est dite #**P**-*complète* si toute fonction $g \in \#\mathbf{P}$ se réduit à f, c'est-à-dire s'il existe :

1. un traducteur $R : \Sigma^* \to \Sigma^*$ qui, étant donnée une instance x de g, produit en temps polynomial une instance $R(x)$ de f, et
2. une fonction $S : \Sigma^* \times \mathbf{N} \to \mathbf{N}$ calculable en temps polynomial qui étant donné x et $f(R(x))$, calcule $g(x)$, c'est-à-dire telle que :

$$\forall x \in \Sigma^*, g(x) = S(x, f(R(x))).$$

Autrement dit, un oracle pour f permet de calculer g en temps polynomial.

Les versions dénombrement de la plupart des problèmes **NP**-complets connus sont #**P**-complètes[1] (la section 28.4 présente une exception à cette « règle »). Il est intéressant de noter que, en dehors d'une poignée d'exceptions, c'est également le cas pour les problèmes **P**. Ceci soulève la question du dénombrement approximatif, en temps polynomial, des solutions des

[1] En fait, la réduction polynomiale utilisée pour réduire un problème **NP**-complet à un autre, met typiquement en relation les solutions de l'un avec celles de l'autre, et préserve ainsi le nombre de solutions ; dans ce cas, la preuve de #**P**-complétude s'obtient directement à partir de la preuve de **NP**-complétude.

problèmes **P** (l'exercice 28.3 étudie le dénombrement approximatif des solutions des problèmes **NP**-complets). Pour ces problèmes, il n'y a que deux situations possibles : soit ils admettent une approximation avec une précision arbitraire, ou bien ils n'admettent aucune approximation (voir section 28.4). La première situation est celle des *schémas d'approximation randomisés totalement polynomiaux*,[2] FPRAS en abrégé.

Considérons un problème **P** dont la fonction de dénombrement f est #**P**-complète. Un algorithme $\mathcal{A}$ est un FPRAS pour ce problème si pour toute instance $x \in \Sigma^*$, et toute erreur $\varepsilon > 0$,

$$\mathbf{Pr}[\,|\mathcal{A}(x) - f(x)| \leqslant \varepsilon f(x)] \geqslant \frac{3}{4}$$

et si le temps de calcul de $\mathcal{A}$ est polynomial en $|x|$ et $1/\varepsilon$ (l'exercice 28.1 présente une méthode pour réduire la probabilité d'erreur d'un FPRAS).

28.1 Dénombrement des solutions DNF

Problème 28.1 (Dénombrement des solutions DNF)[3] Soit $f = C_1 \vee C_2 \vee \cdots \vee C_m$ une formule sous forme normale disjonctive, avec n variables booléennes $x_1, \ldots, x_n$. Chaque clause C_i est de la forme $C_i = l_1 \wedge l_2 \wedge \cdots \wedge l_{r_i}$, où chaque l_j est un littéral, c'est-à-dire une variable booléenne ou sa négation. Sans perte de généralité, nous pouvons supposer que chacune des clauses est satisfaisable, c'est-à-dire ne contient pas à la fois une variable et sa négation. Le problème est de calculer $\#f$, le nombre d'instanciations des variables qui satisfont f.

L'idée principale est de construire une variable aléatoire X, facile à échantillonner, qui soit un estimateur non biaisé de $\#f$, c'est-à-dire telle que $\mathbf{E}[X] = \#f$. Si en plus, l'écart-type de X est à un facteur polynomial de $\mathbf{E}[X]$, alors nous obtiendrons directement un FPRAS pour $\#f$, en renvoyant la moyenne d'un nombre polynomial (en n et $1/\varepsilon$) de tirages de X.

La construction d'un estimateur non biaisé de $\#f$ est aisée. Il suffit de prendre une variable aléatoire uniforme Y sur l'ensemble des 2^n instanciations, et de définir $Y(\tau) = 2^n$ si τ satisfait f, et $Y(\tau) = 0$ sinon (voir exercice 28.4). Cependant, l'écart-type de cette variable aléatoire peut être énorme, et ne donne pas de FPRAS. Supposons par exemple que f n'admette qu'un nombre polynomial d'instanciations positives. Alors au bout d'un nombre polynomial de tirages, Y vaudra 0 tout le temps avec forte probabilité, donnant ainsi une piètre estimation de $\#f$.

Nous corrigeons ce défaut en définissant une variable aléatoire qui prend ses valeurs parmi les instanciations qui satisfont au moins une clause de f.

[2] *Fully polynomial randomized approximation scheme (FPRAS)*, en anglais.
[3] *Counting DNF solutions*, en anglais.

Notons S_i l'ensemble des instanciations $x_1, \ldots, x_n$ qui satisfont la clause C_i. Clairement, $|S_i| = 2^{n-r_i}$, où r_i est le nombre de littéraux de la clause C_i. Remarquons que $\#f = |\cup_{i=1}^{m} S_i|$. Notons $c(\tau)$ le nombre de clauses que l'instanciation τ satisfait. Soit M le multi-ensemble défini par l'union des (S_i) ; M contient chaque instanciation τ, $c(\tau)$ fois. Remarquons que $|M| = \sum_i |S_i|$ est facile à calculer.

Définissons la variable aléatoire X qui sélectionne une instanciation τ avec probabilité $c(\tau)/|M|$, puis renvoie la valeur $X(\tau) = |M|/c(\tau)$. Commençons par démontrer que X s'échantillonne efficacement, c'est-à-dire en temps polynomial.

Lemme 28.2 *La variable aléatoire X s'échantillonne efficacement.*

Preuve : Tirer un élément uniformément dans le multi-ensemble M permet de sélectionner une instanciation selon la loi souhaitée. Pour le faire en temps polynomial, il suffit de commencer par tirer une clause C_i au hasard avec probabilité $|S_i|/|M|$, puis de tirer uniformément dans S_i, une instanciation qui satisfait C_i.

La probabilité qu'une instanciation τ soit tirée est alors :

$$\sum_{i\,:\,\tau \text{ satisfait } C_i} \frac{|S_i|}{|M|} \times \frac{1}{|S_i|} = \frac{c(\tau)}{|M|}.$$

□

Lemme 28.3 *X est un estimateur non biaisé de $\#f$.*

Preuve :

$$\mathbf{E}[X] = \sum_{\tau} \mathbf{Pr}[\tau \text{ est sélectionné}] \cdot X(\tau) = \sum_{\tau \text{ satisfait } f} \frac{c(\tau)}{|M|} \times \frac{|M|}{c(\tau)} = \#f.$$

□

Les valeurs de X appartiennent à un intervalle « d'extension polynomiale », c'est-à-dire dont le *ratio* des extrémités est borné par un polynôme. Ainsi l'écart-type de X ne peut être très grand devant son espérance. C'est l'objet du lemme suivant, qui nous donnera le FPRAS.

Lemme 28.4 *Soit m le nombre de clauses de f, alors*

$$\frac{\sigma(X)}{\mathbf{E}[X]} \leqslant m - 1.$$

Preuve : Posons $\alpha = |M|/m$. Clairement, $\mathbf{E}[X] \geqslant \alpha$. Pour chaque instanciation τ de f, nous avons $1 \leqslant c(\tau) \leqslant m$. Ainsi, $X(\tau)$ appartient à l'intervalle $[\alpha, m\alpha]$; sa déviation par rapport à sa moyenne est donc inférieure à $(m-1)\alpha$, c'est-à-dire $|X(\tau) - \mathbf{E}[X]| \leqslant (m-1)\alpha$. Ainsi, l'écart-type de X est borné par $(m-1)\alpha \leqslant (m-1)\mathbf{E}[X]$. CQFD. □

Enfin, nous allons démontrer que renvoyer la moyenne d'un nombre polynomial (en n et $1/\varepsilon$) de tirages de X donne un FPRAS pour $\#f$. Soit X_k la variable aléatoire correspondant à la moyenne de k tirages indépendants de X.

Lemme 28.5 *Pour tout $\varepsilon > 0$,*

$$\mathbf{Pr}[\,|X_k - \#f| \leqslant \varepsilon \#f\,] \geqslant 3/4,$$

pour tout $k \geqslant 4(m-1)^2/\varepsilon^2$.

Preuve : Appliquons l'inégalité de Tchebycheff (voir section B.2), en prenant $a = \varepsilon \mathbf{E}[X_k]$. Pour les valeurs k indiquées, nous obtenons :

$$\mathbf{Pr}[\,|X_k - \mathbf{E}[X_k]| \geqslant \varepsilon \mathbf{E}[X_k]\,] \leqslant \left(\frac{\sigma(X_k)}{\varepsilon \mathbf{E}[X_k]}\right)^2 = \left(\frac{\sigma(X)}{\varepsilon\sqrt{k}\mathbf{E}[X]}\right)^2 \leqslant \frac{1}{4}.$$

L'égalité résulte des égalités $\mathbf{E}[X_k] = \mathbf{E}[X]$ et $\sigma(X_k) = \sigma(X)/\sqrt{k}$, et la dernière inégalité du lemme 28.4, CQFD. □

Théorème 28.6 *Il existe un FPRAS pour le problème du dénombrement des solutions DNF.*

28.2 Fiabilité d'un réseau

Problème 28.7 (Fiabilité d'un réseau)[4] Étant donné un graphe connexe non orienté $G = (V, E)$, et une probabilité de panne p_e pour chaque arête e, calculer la probabilité que le graphe soit déconnecté.

Le graphe G est déconnecté si toutes les arêtes d'une coupe quelconque $(C, \overline{C})$, $C \subsetneq V$, sont en panne. Nous donnons ici un FPRAS pour ce problème.

Commençons par le cas où toutes les arêtes ont la même probabilité p de tomber en panne. Nous autorisons cependant l'existence d'arêtes multiples entre chaque paire de sommets dans G. Notons PANNE(p) la probabilité que G soit déconnecté. Si PANNE(p) est minorée par un polynôme en n, elle peut être efficacement estimée par un échantillonnage de Monte Carlo (voir preuve du théorème 28.11 pour plus de détails). Traitons donc le cas difficile où PANNE(p) est petit. Supposons que PANNE(p) $\leqslant n^{-4}$. Le choix de cette borne s'expliquera par la suite.

La probabilité qu'une coupe donnée $(C, \overline{C})$ soit totalement déconnectée est simplement p^c, où c est le nombre d'arêtes de la coupe. Comme la probabilité de panne d'une coupe décroît exponentiellement avec sa capacité,

[4] *Network reliability*, en anglais.

les coupes les plus importantes pour estimer PANNE(p) sont les coupes de « petite » capacité. L'algorithme est construit autour de deux idées :

1. Nous allons montrer que pour tout $\varepsilon > 0$, un nombre polynomial (en n et $1/\varepsilon$) de « petites » coupes seulement est responsable d'une fraction $1-\varepsilon$ de la probabilité de panne globale PANNE(p), et qu'on peut de plus énumérer en temps polynomial ces coupes, notons-les $E_1, \ldots E_k \subseteq E$.
2. Nous allons construire une formule DNF f de taille polynomiale dont la probabilité d'être satisfaite est exactement la probabilité qu'une de ces coupes (au moins) soit totalement déconnectée.

Une conséquence du premier point est qu'il suffirait d'estimer la probabilité qu'une des coupes $E_1, \ldots, E_k$ soit totalement déconnectée. Mais ceci est non trivial du fait des corrélations. La seconde idée est donc de réduire ce problème au dénombrement de solutions DNF, pour lequel nous avons un FPRAS.

Nous associons une variable booléenne x_e à chaque arête e. x_e vaut vrai avec probabilité p, la probabilité de panne de l'arête e. Nous associons à chaque coupe $E_i = \{e_1, \ldots, e_j\}$, la clause $D_i = x_{e_1} \wedge \cdots \wedge x_{e_j}$, la conjonction des variables associées aux arêtes de E_i. La probabilité que cette clause soit satisfaite, est exactement la probabilité de déconnexion de la coupe E_i. Enfin, soit $f = D_1 \vee \cdots \vee D_k$ la disjonction des clauses associées aux coupes.

28.2.1 Majoration du nombre de coupes presque minimum

La première idée repose sur le fait qu'on sait majorer le nombre des coupes de capacité minimum ou presque minimum dans un graphe non orienté. Notons c la capacité d'une coupe minimum de G. Rappelons que toutes les arêtes de G ont une capacité unitaire, et que des arêtes multiples sont autorisées dans G.

Lemme 28.8 *Le nombre de coupes minimum de $G = (V, E)$ est inférieur à $n(n-1)/2$.*

Preuve : Nous appelons *contraction* de l'arête uv d'un graphe, l'opération qui consiste à fusionner les deux sommets u et v. Toutes les arêtes entre u et v sont éliminées, et toutes celles issues de u ou v sont redirigées vers le nouveau sommet fusionné – leur nombre est conservé.

Considérons à présent le processus suivant de *contraction randomisée*. Sélectionner une arête aléatoire uv dans le graphe courant, contracter cette arête et répéter jusqu'à qu'il ne reste plus que deux sommets. Notons S et $V - S$, les ensembles de sommets correspondant aux deux derniers sommets à la fin de la procédure de contraction. L'algorithme renvoie la coupe $(S, \overline{S})$. Nous dirons que cette coupe a *survécu*. Clairement, une coupe survit ssi aucune de ces arêtes n'est sélectionnée durant l'exécution de l'algorithme.

Soit $(C, \overline{C})$ une coupe minimum de G. Démontrons que :

$$\mathbf{Pr}[(C,\overline{C}) \text{ survit}] \geqslant \frac{1}{\binom{n}{2}}.$$

Cette minoration entraînera le lemme par un argument astucieux. Notons M le nombre de coupes minimum de G. Les survies de chacune de ces coupes sont des événements mutuellement exclusifs, et la probabilité totale de ces événements est inférieure à 1. Par conséquent, $M/(n(n-1)/2) \leqslant 1$, CQFD.

Considérons une étape quelconque du processus de contraction randomisée, et notons H le graphe au début de cette étape. Comme toute coupe du graphe contracté définit une coupe de même capacité dans le graphe initial, la capacité de toute coupe de H est supérieure à c. C'est en particulier le cas pour les coupes séparant un sommet du reste de H. Ainsi, le degré de tout sommet de H est supérieur à c. H a donc plus de $cm/2$ arêtes, où m est le nombre de sommets de H.

La probabilité conditionnelle que la coupe $(C,\overline{C})$ survive à l'itération courante, sachant qu'elle a survécu jusqu'à présent, est donc supérieure à : $(1 - \frac{c}{cm/2}) = (1 - 2/m)$: il s'agit simplement de la probabilité que l'arête ne soit pas choisie dans la coupe $(C,\overline{C})$. Comme la probabilité que $(C,\overline{C})$ survive à l'exécution complète de l'algorithme est égale au produit de ces probabilités conditionnelles,

$$\mathbf{Pr}[(C,\overline{C}) \text{ survit}] \geqslant \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) = \frac{1}{\binom{n}{2}}. \qquad \square$$

Étant donné $\alpha \geqslant 1$, nous dirons qu'une coupe est *α-minimum* si sa capacité est inférieure à αc.

Lemme 28.9 *Pour tout $\alpha \geqslant 1$, le nombre de coupes α-minimum de G est inférieur à $n^{2\alpha}$.*

Preuve : Démontrons le lemme pour α demi-entier. La preuve pour α quelconque repose sur les mêmes idées appliquées aux coefficients binomiaux généralisés. Posons $k = 2\alpha$ ($k \in \mathbb{N}$).

Considérons le processus en deux temps suivant : lancer l'algorithme de contraction randomisée jusqu'à ce qu'il ne reste plus que k sommets ; puis, tirer une coupe aléatoire uniformément parmi les 2^{k-1} coupes du graphe résiduel. La coupe obtenue définit une coupe dans le graphe initial.

Soit $(C,\overline{C})$ une coupe α-minimum quelconque de G. Démontrons que la probabilité qu'elle survive aux deux phases de l'algorithme est supérieure à $1/n^{2\alpha}$, ce qui conclura la preuve (comme précédemment).

Notons H le graphe au début d'une étape quelconque de la première phase. De même qu'au lemme 28.8, si H a m sommets, il contient plus de $mc/2$ arêtes. Ainsi, la probabilité conditionnelle que $(C,\overline{C})$ survive à l'itération courante, sachant qu'elle survécu jusqu'à présent, est supérieure à : $1 - \frac{\alpha c}{mc/2} = 1 - 2\alpha/m$. La probabilité que $(C,\overline{C})$ survive à la première phase de l'algorithme est donc supérieure à :

$$\left(1-\frac{2\alpha}{n}\right)\left(1-\frac{2\alpha}{n-1}\right)\cdots\left(1-\frac{2\alpha}{k+1}\right)=\frac{1}{\binom{n}{k}}.$$

La probabilité conditionnelle que $(C,\overline{C})$ survive à la deuxième phase, étant donné qu'elle a survécu à la première, est $1/2^{k-1}$. Ainsi,

$$\mathbf{Pr}[(C,\overline{C}) \text{ survit aux deux phases}] \geqslant \frac{1}{\binom{n}{k}2^{k-1}} \geqslant \frac{1}{n^k} = \frac{1}{n^{2\alpha}}.$$

□

28.2.2 Analyse

Rappelons que nous étudions le cas où PANNE$(p) \leqslant n^{-4}$. Voici le pourquoi de ce choix. La probabilité de déconnexion totale d'une coupe minimum vaut $p^c \leqslant$ PANNE$(p) \leqslant n^{-4}$. Réécrivons $p^c = n^{-(2+\delta)}$, avec $\delta \geqslant 2$. D'après le lemme 28.9, pour tout $\alpha \geqslant 1$, la probabilité de déconnexion totale de toutes les coupes de capacité inférieure à αc est inférieure à $p^{\alpha c}n^{2\alpha} = n^{-\alpha\delta}$. Cette chute rapide de la probabilité de déconnexion totale de toutes les coupes de capacité inférieure à αc nous permet de borner celle des coupes de « grande » capacité.

Lemme 28.10 *Pour tout $\alpha \geqslant 1$,*

$$\mathbf{Pr}[\textit{une coupe de capacité} > \alpha c \textit{ est tot. déconnectée}] \leqslant n^{-\alpha\delta}\left(1+\frac{2}{\delta}\right).$$

Preuve : Numérotons les coupes de G par capacité croissante. Notons respectivement c_k et p_k, la capacité et la probabilité de déconnexion totale de la k-ième coupe dans cet ordre. Soit a le numéro de la première coupe de capacité supérieure strictement à αc. Il suffit de démontrer que :

$$\sum_{k\geqslant a} p_k \leqslant n^{-\alpha\delta}\left(1+\frac{2}{\delta}\right).$$

Coupons la somme à gauche en deux morceaux. Commençons par les $n^{2\alpha}$ premiers termes. Chacun de ces termes est inférieur à $p^{\alpha c} = n^{-\alpha(2+\delta)}$. Ainsi, leur somme est inférieure à $n^{-\alpha\delta}$.

Bornons maintenant le reste de la somme. Clairement, cette somme est inférieure à $\sum_{k>n^{2\alpha}} p_k$. D'après le lemme 28.9, il existe moins de $n^{2\alpha}$ coupes de capacité inférieure à αc. Ainsi, pour tout β, $c_{n^{2\beta}} \geqslant \beta c$. Posons $k = n^{2\beta}$, alors $c_k \geqslant c \ln k/2 \ln n$, et :

$$p_k = p^{c_k} \leqslant (p^c)^{\frac{\ln k}{2\ln n}} = k^{-(1+\delta/2)}.$$

Finalement,

$$\sum_{k>n^{2\alpha}} p_k \leqslant \int_{n^{2\alpha}}^{\infty} k^{-(1+\delta/2)} \mathrm{d}k \leqslant \frac{2}{\delta}\, n^{-\alpha\delta}.$$

CQFD. □

Théorème 28.11 *Il existe un FPRAS pour estimer la fiabilité d'un réseau.*

Preuve : Commençons par traiter le cas où toutes les arêtes du graphe G ont la même probabilité de panne p.

Si $\text{PANNE}(p) > n^{-4}$, nous utilisons un échantillonnage de Monte Carlo. Tirer à pile-ou-face avec un biais p la panne de chaque arête, et vérifier si le graphe G est connecté. Répéter indépendamment $\Theta(\log n/(\varepsilon^2\,\text{PANNE}(p)))$ fois ce tirage, c'est-à-dire $O(n^4 \log n/\varepsilon^2)$ fois, et renvoyer le nombre moyen de fois que G était déconnecté. Une application directe de la borne de Chernoff démontre que cette moyenne appartient à $[(1-\varepsilon)\,\text{PANNE}(p), (1+\varepsilon)\,\text{PANNE}(p)]$ avec forte probabilité.

Supposons à présent que $\text{PANNE}(p) \leqslant n^{-4}$. Pour tout $\varepsilon > 0$, nous souhaitons déterminer un α tel que la probabilité de la déconnexion totale d'une des coupes de capacité $> \alpha c$ soit inférieure à $\varepsilon\,\text{PANNE}(p)$. D'après le lemme 28.10, il suffit de prendre α tel que :

$$n^{-\alpha\delta}\left(1+\frac{2}{\delta}\right) \leqslant \varepsilon\,\text{PANNE}(p) \leqslant \varepsilon n^{-(2+\delta)}.$$

En résolvant cette équation, nous trouvons :

$$\alpha = 1 + \frac{2}{\delta} - \frac{\ln(\varepsilon)}{\delta \ln n} \leqslant 2 - \frac{\ln(\varepsilon)}{2 \ln n}.$$

D'après le lemme 28.9, $c_{n^{2\alpha}} > \alpha c$. Ainsi pour la valeur de α ci-dessus,

$$\mathbf{Pr}[\text{une des } n^{2\alpha} \text{ premières coupes est tot. déconnectée}] \geqslant (1-\varepsilon)\,\text{PANNE}(p).$$

Les $n^{2\alpha} = O(n^4/\varepsilon)$ plus petites coupes s'énumèrent en temps polynomial (voir exercices). Nous pouvons donc construire la formule DNF correspondant à cet événement, et estimer la probabilité qu'elle soit satisfaite, comme nous l'avons vu précédemment.

Nous terminons cette preuve en démontrant comment « réduire » le cas de probabilités de panne d'arête arbitraires au cas uniforme étudié ci-dessus. Notons p_e la probabilité de panne de l'arête e. Fixons un paramètre θ petit. Remplaçons l'arête e par $k_e = -(\ln p_e)/\theta$ arêtes parallèles de probabilités de panne identiques $1-\theta$. Alors, la probabilité que ces k_e arêtes tombent en panne simultanément est :

$(1-\theta)^{-(\ln p_e)/\theta}$.

Quand $\theta \to 0$, cette probabilité converge vers p_e. Notons H le graphe obtenu en appliquant cette transformation sur chaque arête de G. À la limite quand $\theta \to 0$, chaque coupe de H a la même probabilité de panne que dans G.

Voici une réalisation efficace de cette idée. Nous ne cherchons en fait qu'à lister les coupes de « petite » capacité de G. Cette liste obtenue, nous pouvons appliquer l'algorithme de l'exercice 28.5 qui dénombre les solutions DNF dans le cas plus général, où chaque variable vaut vrai avec une probabilité propre p_e. Observez que faire varier θ modifie les capacités des coupes de H sans changer leurs valeurs relatives. Ainsi, il suffit de donner un poids $-\ln p_e$ à chaque arête e de G, et de trouver les coupes de « petite » capacité dans ce graphe. CQFD. □

28.3 Exercices

28.1 Étant donné un FPRAS pour un problème, démontrez que sa probabilité de réussite peut être portée à $1-\delta$, pour tout $\delta > 0$, en augmentant le temps de calcul d'un facteur $O(\log(1/\delta))$ seulement.
Indication : Exécutez le FPRAS $O(\log(1/\delta))$ fois et renvoyez la valeur médiane.

28.2 Supposez qu'on donne une définition plus exigeante d'un FPRAS en demandant que l'erreur soit inférieure à une constante additive fixée α avec forte probabilité, c'est-à-dire

$$\mathbf{Pr}[f(x) - \alpha \leqslant A(x) \leqslant f(x) + \alpha] \geqslant \frac{3}{4}.$$

Démontrez que s'il existe un tel algorithme pour un problème #**P**-complet, alors **P**= **NP**.

28.3 Montrez que s'il existe un FPRAS pour dénombrer les instanciations satisfaisant SAT, alors tout problème de **NP** se résout en temps polynomial randomisé. Quelles sont les plus faibles conditions à imposer à cet algorithme de dénombrement pour SAT pour que son existence entraîne cette conséquence ? Qu'est-ce que cela implique sur le problème du dénombrement approximatif des solutions des autres problèmes **NP**-complets ?
Indication : Augmentez le nombre de solutions par amplification : étant donné une formule SAT f, prenez une formule f' sur k nouvelles variables qui soit une tautologie ; puis remarquez que le nombre de solutions de $\phi = f \wedge f'$ est $\#f \cdot 2^k$.

28.4 Étant donné une formule DNF f, soit Y une variable aléatoire qui, étant donné une instance aléatoire τ, vaut 2^n si τ satisfait f et 0 sinon. Montrez que Y est un estimateur non biaisé de $\#f$. Quelle est la valeur maximale du rapport $\sigma(Y)/\mathbf{E}[Y]$?

28.5 (Karp et Luby [173]) Soient une formule DNF f sur n variables booléennes $x_1, \ldots, x_n$, et des probabilités $p_1, \ldots, p_n$ avec lesquelles chacune de ces variables vaut vrai (indépendamment). Notons D la loi de probabilité correspondante sur les 2^n instanciations possibles, et p la probabilité que f soit satisfaite par une instanciation aléatoire tirée suivant D. Proposez un FPRAS pour estimer p.
Indication : Notez q_i la probabilité que la clause C_i soit satisfaite par une instanciation aléatoire tirée suivant D, et $Q = \sum_i q_i$. Puis, considérez la variable aléatoire X qui tire une instanciation τ avec probabilité $\mathbf{Pr}_D[\tau]\, c(\tau)/Q$, et renvoie $X(\tau) = Q/c(\tau)$.

28.6 Un *générateur uniforme* d'un problème **NP** Π est un algorithme randomisé polynomial $\mathcal{A}$ qui, étant donné une instance I d'un problème, renvoie soit une solution de I, soit le symbole spécial $\perp$, de sorte que :

- chaque solution de I soit obtenue avec la même probabilité, c'est-à-dire il existe $\alpha \in (0, 1]$ tel que

$$\mathbf{Pr}[\mathcal{A} \text{ renvoie } s] = \alpha, \text{ pour toute solution } s \text{ de } I, \text{ et}$$

- la probabilité de renvoyer $\perp$, c'est-à-dire la probabilité d'échec, soit $< 1/2$.

Exhibez un générateur uniforme pour tirer une instanciation aléatoire qui satisfasse une formule DNF donnée.
Indication : Les idées principales résident dans la construction de la variable aléatoire X.

28.7 (Jerrum, Valiant, et Vazirani [155]) Soit Π un problème **NP** autoréductible (voir section A.5 et exercice 1.15). Démontrez qu'il existe un FPRAS pour Π ssi il admet un générateur quasi uniforme. Un *générateur quasi uniforme* pour Π est un algorithme randomisé polynomial $\mathcal{A}$ tel que pour tout $\mu > 0$ et toute instance I de Π, il existe $\alpha \in (0, 1]$ tel que :

- pour tout solution s de I, $\mathbf{Pr}[\mathcal{A} \text{ renvoie } s] \in [(1-\mu)\alpha, (1+\mu)\alpha]$,
- $\mathbf{Pr}[\mathcal{A}$ échoue et ne renvoie pas de solution$] < 1/2$, et
- le temps d'exécution de $\mathcal{A}$ est polynomial en $|I|$ et $\log(1/\mu)$.

Observez que contrairement à un FPRAS, qui ne peut réaliser qu'une erreur polynomialement petite, un générateur uniforme peut engendre une erreur exponentiellement petite (μ) en temps polynomial.
Indication : Tout d'abord, construisez un générateur uniforme en supposant que le FPRAS ne fait pas d'erreur — parcourez l'arbre d'autoréductibilité de I avec des biais déterminés par les estimations du nombre de solutions ; puis,

acceptez une feuille avec une probabilité convenable pour assurer l'uniformité. Utilisez le fait qu'on peut réduire exponentiellement la probabilité d'erreur du FPRAS pour obtenir un générateur uniforme. Pour la contraposée, exhibez une instance I_α, avec $|I_\alpha| < |I|$, et une bonne estimation du ratio des nombres de solutions de I et de I_α.

28.8 (Jerrum, Valiant, et Vazirani [155]) Cet exercice tend à montrer que le dénombrement du nombre de circuits simples d'un graphe orienté est fondamentalement inapproximable. Démontrez que s'il existe un générateur quasi uniforme pour ce problème, alors il existe un algorithme randomisé polynomial qui décide si un graphe orienté admet un circuit hamiltonien.
Indication : Construisez un graphe G' qui amplifie le nombre de circuits de chaque longueur de G. Notez que les circuits doivent être d'autant plus amplifiés qu'ils sont longs, de sorte que la plupart des circuits de G' soient de longueur maximum et correspondent aux plus grands circuits de G.

28.9 Montrez comment utiliser l'algorithme de contraction randomisée du lemme 28.8 pour trouver une coupe minimum dans un graphe non orienté.

28.10 (Karger et Stein [167]) Proposez un algorithme randomisé qui énumère toutes les coupes α-minimum de G en utilisant l'algorithme de contraction et le lemme 28.9.

Les trois exercices suivants (de Vazirani et Yannakakis [261]) construisent un algorithme déterministe, de *délai polynomial*, qui énumère toutes les coupes d'un graphe non orienté par capacité croissante, c'est-à-dire tel qu'il s'écoule un temps polynomial entre les sorties de deux coupes successives.

Donnons-nous un graphe $G = (V, E)$ ayant n sommets numérotés de 1 à n ainsi que deux sommets supplémentaires notés s et t. Toute coupe entre s et t de G se représente par un vecteur de n bits. On peut représenter une coupe partiellement définie, dans laquelle seuls les côtés des k premiers sommets sont déterminés, par un vecteur de k bits. Considérons un arbre binaire T de hauteur n, dont les feuilles représentent les coupes entre s et t et les nœuds internes, les coupes définies partiellement. Étant donné une coupe définie partiellement, toutes les coupes qui lui sont consistantes appartiennent au sous-arbre associé. Un simple calcul de flot maximum permet de trouver une coupe de poids minimum de ce sous-arbre.

28.11 Soit $\boldsymbol{a}$ un vecteur de $n - k$ bits représentant aussi bien une coupe partiellement définie que le nœud interne associé dans T. Le sous-arbre T' enraciné en $\boldsymbol{a}$ est de hauteur k et contient 2^k feuilles (2^k coupes entre s et t). Parmi ces 2^k coupes, soit $\boldsymbol{a}'$ une coupe de poids minimum. Démontrez que l'ensemble des $2^k - 1$ coupes restantes de T' peut être partitionné en k sous-arbres de hauteurs $0, 1, \ldots, k - 1$.

28.12 Proposez un algorithme qui utilise un tas pour énumérer toutes les coupes entre s et t de G par poids croissant.
Indication : Le tas est initialement réduit à une coupe minimum de G. À chaque instant, l'ensemble des coupes qui n'ont pas encore été énumérées, se partitionne en sous-arbres (voir exercice 28.11). Le tas contient une coupe minimum de chacun des sous-arbres.

28.13 Proposez un algorithme de délai polynomial qui énumère toutes les coupes d'un graphe non orienté par poids croissant.
Indication : distinguez un sommet s du graphe qui sera toujours du côté 0 de coupe, et numérotez les autres sommets de 1 à n. Une coupe est déterminée par un vecteur de n bits spécifiant les côtés des sommets numérotés de 1 à n. L'algorithme diffère dans la recherche d'une coupe minimum dans le sous-arbre enraciné en un nœud interne $0^k, k < n$. Il s'agit de trouver une coupe minimum séparant les sommets $s, 1, \ldots, i$ du sommet $i+1$ pour $k \leqslant i < n$, et de sélectionner la plus légère de ces coupes.

28.14 (Karger [164]) Considérons la généralisation suivante du problème de l'évaluation de la fiabilité d'un réseau, où il s'agit d'estimer la probabilité que G se déconnecte en au moins r composantes connexes, où r est une constante fixée. Proposez un FPRAS pour ce problème.

28.4 Notes

La classe de dénombrement $\#\mathbf{P}$ a été définie par Valiant [258]. Le FPRAS qui dénombre les solutions DNF est dû à Karp et Luby [173], qui introduisirent par là même la notion de FPRAS (voir également Karp, Luby et Madras [174]). Le FPRAS qui évalue la fiabilité d'un réseau est dû à Karger [164].

La plupart des algorithmes de dénombrement approximatif reposent sur la construction d'un générateur uniforme pour le problème, en utilisant l'équivalence établie à l'exercice 28.7. Broder [35] a introduit l'usage de chaînes de Markov rapidement mélangeantes pour l'obtention de générateurs quasi uniformes (voir également Mihail [214]).

Jerrum et Sinclair [153] ont donné le premier FPRAS utilisant cette approche, pour dénombrer les couplages parfaits d'un graphe biparti dense (dont tous les sommets sont de degré $\geqslant n/2$; voir aussi section 30.3). Ils ont démontré également qu'un compteur grossièrement approximatif, d'erreur polynomiale, peut être transformé en un FPRAS (d'erreur polynomialement petite), en définissant une chaîne de Markov adéquate fondée sur l'arbre d'autoréductibilité de l'instance. Il s'ensuit qu'un problème $\#\mathbf{P}$-complet admet un FPRAS ou bien est essentiellement inapproximable (voir exercice 28.8). Un exemple de problème **NP**-complet dont le problème de dénombrement associé n'appartient pas à $\#\mathbf{P}$, à moins que $\mathbf{P} = \mathbf{NP}$, est donné dans Durand, Hermann et Kolaitis [69] (ils y établissent que ce problème de dénombrement

est en fait complet pour une classe plus grande : #**NP**). Se reporter à Jerrum et Sinclair [150], Sinclair [251] et aux références de la section 30.3 pour des algorithmes de dénombrement à base de chaînes de Markov.

29 Difficulté de l'approximation

Une des réalisations théoriques les plus remarquables de l'algorithmique exacte est d'avoir caractérisé presque complètement[1] la complexité intrinsèque des principaux problèmes informatiques naturels, *modulo* quelques conjectures largement admises. Des avancées récentes font penser que nous aurons également, un jour, une compréhension de l'approximabilité des problèmes d'optimisation **NP**-difficiles. Ce chapitre propose un bref état de l'art de ces résultats.

On peut regrouper les résultats actuels sur la difficulté de l'approximation en trois catégories principales. Pour les problèmes de minimisation, les facteurs d'approximation limites sont, suivant la classe, constants (> 1), ou $\Omega(\log n)$, ou n^{ε} pour un $\varepsilon > 0$ constant donné, où n est la taille de l'instance. Pour les problèmes de maximisation, ces facteurs sont constants (< 1), ou $O(1/\log n)$, ou $1/n^{\varepsilon}$ pour un $\varepsilon > 0$ constant. Ce chapitre présente des résultats sur la difficulté de l'approximation de MAX-3SAT, de la couverture par sommets et de l'arbre de Steiner pour la première classe, de la couverture par ensembles pour la seconde, et de la clique maximum pour la troisième. Nous caractérisons ici la difficulté de l'approximation des versions non pondérées de ces problèmes (c'est-à-dire avec des coûts unitaires), et donc leur « inapproximabilité intrinsèque ».

29.1 Réductions, écart et facteur d'approximation limites

Commençons par rappeler la terminologie utilisée pour établir l'impossibilité de la résolution exacte des problèmes d'optimisation. L'outil principal est le théorème de Cook et Levin, qui établit qu'à moins que $\mathbf{P} = \mathbf{NP}$, il est impossible de distinguer en temps polynomial les instances satisfaisables de SAT des autres. Pour prouver l'impossibilité de résoudre exactement le problème de la couverture par sommets (par exemple), on démontre *via* une réduction polynomiale à partir de SAT, qu'il est impossible de distinguer, en temps polynomial, les graphes qui admettent une couverture par sommets de

[1] À quelques exceptions (importantes) près, comme le problème de l'isomorphisme entre deux graphes.

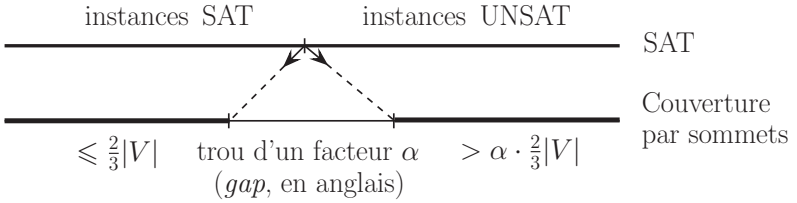
taille k des autres, où k est une constante donnée en entrée avec le graphe, à moins que $\mathbf{P} = \mathbf{NP}$. Comme un algorithme de résolution exact pourrait faire cette distinction, cette réduction démontre l'inexistence d'un algorithme exact efficace.

L'outil principal des résultats sur la difficulté de l'approximation est le théorème PCP qui sera énoncé section 29.2. On utilise ce théorème pour prouver l'existence d'une réduction polynomiale qui permet d'établir la difficulté de l'approximation d'un problème. Par exemple, pour la couverture par sommets, cette réduction associe à toute instance ϕ de SAT, un graphe $G = (V, E)$ tel que :

- si ϕ est satisfaisable, G admet une couverture par sommets de taille $\leqslant \frac{2}{3}|V|$, et
- si ϕ n'est pas satisfaisable, la taille de la plus petite couverture par sommets de G est $> \alpha \cdot \frac{2}{3}|V|$, où $\alpha > 1$ est une constante fixée.

Fait 29.1 *La réduction ci-dessus implique qu'il n'existe pas d'α-approximation (polynomiale) pour la couverture par sommets, à moins que* $\mathbf{P} = \mathbf{NP}$.

Preuve : La réduction démontre qu'il est impossible (à moins que $\mathbf{P} = \mathbf{NP}$) de distinguer les graphes ayant une couverture par sommets de taille $\leqslant \frac{2}{3}|V|$, de ceux ayant une couverture par sommets de taille $> \alpha \cdot \frac{2}{3}|V|$. Une α-approximation pour la couverture par sommets, trouverait une couverture par sommets de taille $\leqslant \alpha \cdot \frac{2}{3}|V|$ dans un graphe du premier type, et permettrait donc de distinguer ces deux classes, contradiction. □



Cette réduction écarte d'un facteur α, les valeurs objectif optimales possibles des deux classes de graphe (remarquons que si $\alpha = 1$, on retrouve la définition d'une réduction polynomiale classique de SAT vers la couverture par sommets). Définissons formellement la notion de *réduction écartante*.[2] Cette définition est légèrement différente suivant que le problème est un problème de minimisation ou de maximisation. Pour simplifier, nous supposerons que nous réduirons toujours depuis SAT.

Soit Π un problème de minimisation. Une réduction écartante de SAT vers Π est caractérisée par deux fonctions f et α. Étant donné une instance ϕ de SAT, une réduction écartante renvoie, en temps polynomial, une instance x de Π, telle que :

- si ϕ est satisfaisable, $\mathrm{OPT}(x) \leqslant f(x)$, et

[2] *Gap-introducing reduction*, en anglais.

– si ϕ n'est pas satisfaisable, $\mathrm{OPT}(x) > \alpha(|x|) \cdot f(x)$.

Remarquons que f est une fonction de l'instance (par exemple, $\frac{2}{3}|V|$ dans l'exemple ci-dessus), et que α est une fonction de la taille de l'instance. Comme Π est un problème de minimisation, la fonction α vérifie $\alpha(|x|) \geqslant 1$.

Lorsque Π est un problème maximisation, une réduction écartante vérifie :

– si ϕ est satisfaisable, $\mathrm{OPT}(x) \geqslant f(x)$, et

– si ϕ n'est pas satisfaisable, $\mathrm{OPT}(x) < \alpha(|x|) \cdot f(x)$.

Dans ce cas, $\alpha(|x|) \leqslant 1$. L'écart $\alpha(|x|)$ est précisément le facteur d'approximation limite démontré par la réduction.

Ayant obtenu une réduction écartante à partir de SAT (ou de n'importe quel autre problème **NP**-difficile) vers un problème d'optimisation Π_1, nous pouvons prouver la difficulté de l'approximation d'un autre problème d'optimisation Π_2, en proposant un autre type de réduction de Π_1 vers Π_2, appelée *réduction d'écart*.[3] Il y a quatre situations possibles, suivant que Π_1 et Π_2 sont des problèmes de minimisation ou de maximisation. Nous donnons la définition pour le cas où Π_1 et Π_2 sont respectivement des problèmes de minimisation et de maximisation. Les autres cas sont similaires.

Une réduction d'écart Γ de Π_1 vers Π_2 est caractérisée par quatre fonctions f_1, α, f_2, et β. Étant donné une instance x de Π_1, une réduction d'écart calcule en temps polynomial une instance y de Π_2 telle que :

– $\mathrm{OPT}(x) \leqslant f_1(x) \;\Rightarrow\; \mathrm{OPT}(y) \geqslant f_2(y)$,

– $\mathrm{OPT}(x) > \alpha(|x|) f_1(x) \;\Rightarrow\; \mathrm{OPT}(y) < \beta(|y|) f_2(y)$.

Remarquons que x et y sont des instances de deux problèmes différents, et qu'il conviendrait mieux d'écrire $\mathrm{OPT}_{\Pi_1}(x)$ et $\mathrm{OPT}_{\Pi_2}(y)$ au lieu de $\mathrm{OPT}(x)$ et $\mathrm{OPT}(y)$, respectivement. Nous éviterons cependant d'alourdir les notations quand il n'y a pas d'ambiguïté. Comme Π_1 et Π_2 sont respectivement des problèmes de minimisation et de maximisation, $\alpha(|x|) \geqslant 1$ et $\beta(|y|) \leqslant 1$.

La composition d'une réduction écartante avec une réduction d'écart engendre une réduction écartante, pourvu que les paramètres s'accordent. Supposons par exemple qu'en plus de la réduction d'écart Γ ci-dessus, nous ayons obtenu une réduction écartante Γ' de SAT vers Π_1 de paramètres f_1 et α. Alors, la composition de Γ' avec Γ donne une réduction écartante de SAT vers Π_2, de paramètres f_2 et β. Cette réduction composée démontre donc qu'il n'existe pas de $\beta(|y|)$-approximation pour Π_2, à moins que $\mathbf{P} = \mathbf{NP}$. Dans les réductions d'écart qui suivront, nous prendrons soin d'assurer que les paramètres s'accordent bien.

Remarque 29.2

– La valeur de « l'écart » β est, en général, différente de celle de α. Dans un sens, la valeur de l'écart n'est donc pas conservée.

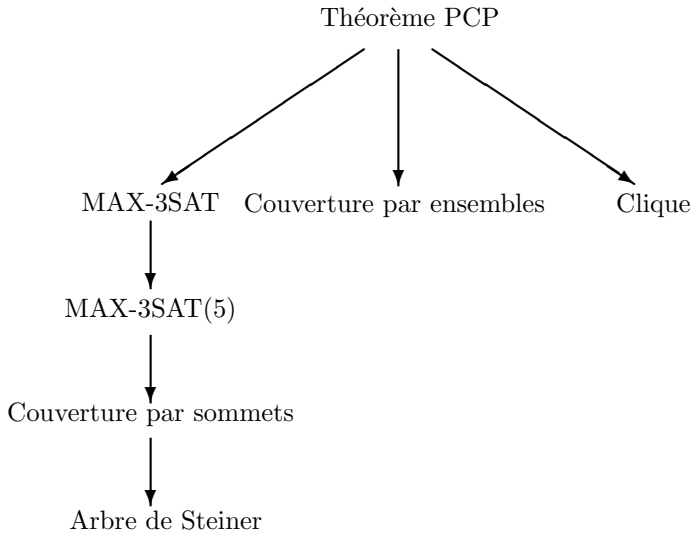
– Aucune contrainte n'est imposée par Γ sur les instances x de Π_1 appartenant au trou[4] de départ, c'est-à-dire telles que $f_1(x) < \mathrm{OPT}(x) \leqslant \alpha(|x|) f_1(x)$.

[3] *Gap-preserving reduction*, en anglais.

[4] *Gap*, en anglais.

– Un algorithme d'approximation pour Π_2 et une réduction d'écart Γ de Π_1 à Π_2 ne donnent pas nécessairement d'algorithme d'approximation pour Π_1, contrairement aux réductions isofacteur (définies à la section A.3.1). Une réduction isofacteur impose en effet de pouvoir transformer toute solution approchée de l'instance réduite y de Π_2 en une solution approchée de x de Π_1.
Mais, Γ couplée avec une réduction écartante adaptée de SAT vers Π_1 suffit pour obtenir un résultat sur la difficulté de l'approximation de Π_2. Les réductions d'écart, moins exigeantes, sont évidemment plus faciles à construire.
– Nous avons déjà présenté quelques réductions écartantes, par exemple aux théorèmes 3.6 et 5.7. Le lecteur est en droit de se demander pourquoi cela ne suffit pas et pourquoi on a dû introduire le théorème PCP. La raison en est que ces deux résultats reposent uniquement sur la possibilité de choisir arbitrairement les poids des arêtes et non sur la structure combinatoire particulière du problème.

La figure suivante résume les réductions d'écart démontrées dans ce chapitre :



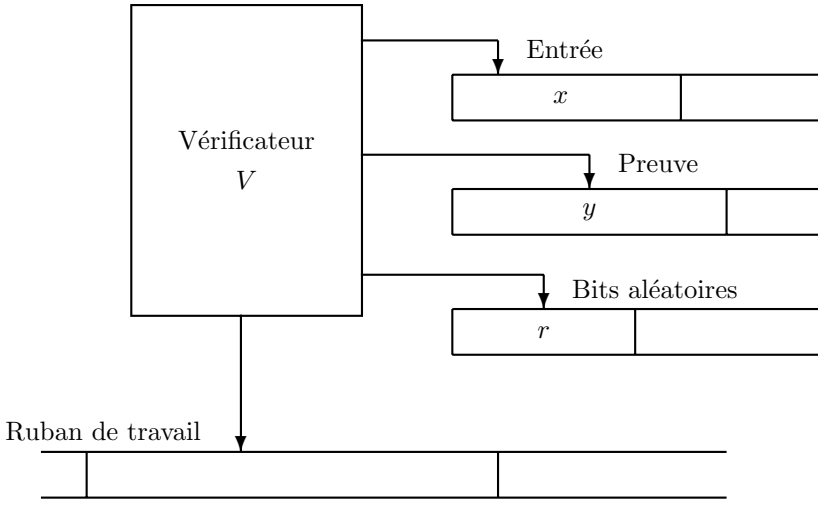
29.2 Le théorème PCP

Les caractérisations probabilistes de la classe **NP** permettent d'obtenir une technique générale pour construire des réductions écartantes. La plus utile de ces caractérisations est donnée par le théorème PCP. PCP signifie *Probabilistically Checkable Proof systems*, c'est-à-dire *système de preuve vérifiable stochastiquement*. Rappelons la définition classique de **NP** (voir annexe A) : **NP** est la classe des langages dont les instances positives admettent

des certificats courts (de longueur polynomiale en la taille de l'entrée) qui peuvent être vérifiés rapidement (en temps polynomial). De manière informelle, une preuve vérifiable stochastiquement pour un langage **NP** encode le certificat de sorte qu'on puisse le vérifier en examinant seulement un très petit nombre de bits choisis aléatoirement.

Un système de preuve vérifiable stochastiquement est caractérisé par deux paramètres : le nombre de bits aléatoires nécessaires au vérificateur, et le nombre de bits du certificat que le vérificateur est autorisé à examiner. Dans cette terminologie, la *preuve* désigne la chaîne codant le certificat. Les plages de valeurs les plus utiles de ces deux paramètres sont respectivement $O(\log n)$ et $O(1)$, ce qui définit la classe $\mathbf{PCP}(\log n, 1)$.

Un *vérificateur* est une machine de Turing en temps polynomial qui dispose de deux rubans supplémentaires, en sus des deux rubans d'entrée et de travail, l'un qui contient une chaîne de bits aléatoires, et l'autre la preuve. La machine peut lire n'importe quel bit de la preuve en spécifiant simplement sa position. Bien entendu, ces positions sont calculées en fonction de l'entrée et de la chaîne de bits aléatoires. À la fin du calcul, la machine passe soit dans un état d'acceptation, soit dans un état de rejet.



Un langage L appartient à $\mathbf{PCP}(\log n, 1)$ s'il existe un vérificateur V, et des constantes c et q, telles que V utilise sur toute entrée x une chaîne de bits aléatoires r de longueur $c \log |x|$ et examine q bits de la preuve, et tel que :

- si $x \in L$, alors il existe une preuve y que V accepte avec probabilité 1,
- si $x \notin L$, alors pour toute preuve y, V accepte y avec une probabilité inférieure à $1/2$,

où les probabilités sont prises sur la chaîne aléatoire r. On appelle *probabilité d'erreur*, la probabilité d'acceptation lorsque $x \notin L$.

Plus généralement, étant donné deux fonctions $r(n)$ et $q(n)$, on définit la classe $\mathbf{PCP}(r(n), q(n))$, où le vérificateur utilise une chaîne de bits

aléatoires de longueur $O(r(n))$ et examine $O(q(n))$ bits de la preuve. Les critères d'acceptation sont les mêmes que ci-dessus. Dans cette terminologie, $\mathbf{NP} = \mathbf{PCP}(0, \text{poly}(n))$, où $\text{poly}(n) = \bigcup_{k \geqslant 0}\{n^k\}$: en effet, le vérificateur n'utilise aucun bit aléatoire et doit accepter les mots du langage et rejeter ceux qui n'en sont pas de façon déterministe, à la seule lecture de la preuve, conformément à la définition de **NP**. Le théorème PCP donne une autre caractérisation de **NP**.

Théorème 29.3 $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

Une des inclusions, $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$, est facile (voir exercice 29.1). L'autre inclusion, $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$, est un résultat difficile et un outil très puissant pour obtenir des résultats sur la difficulté de l'approximation. À ce jour, la preuve connue de ce résultat est trop compliquée pour être exposée ici. Heureusement, l'énoncé du théorème suffit à obtenir les résultats de difficulté.

Afin d'aider le lecteur à comprendre la portée du théorème PCP, faisons l'observation suivante. Il est très facile de construire un vérificateur pour 3SAT dont la probabilité d'erreur (c'est-à-dire la probabilité d'accepter une formule insatisfaisable) est $\leqslant 1 - 1/m$, où m est le nombre de clauses de la formule 3SAT, ϕ, donnée en entrée. Le vérificateur attend pour preuve de ϕ, une instanciation satisfaisante. Il utilise $O(\log m)$ bits aléatoires pour sélectionner une clause aléatoire de ϕ. Il lit les valeurs des trois variables de cette clause – remarquons qu'il ne lit qu'un nombre constant de bits. Puis, il accepte ssi les valeurs de ces variables satisfont la clause. Clairement, si ϕ est satisfaisable, il existe une preuve telle que le vérificateur accepte avec probabilité 1 ; et si ϕ n'est pas satisfaisable, pour toute preuve, le vérificateur n'accepte qu'avec une probabilité $\leqslant 1 - 1/m$. La partie intéressante et difficile du théorème PCP consiste à rendre la probabilité d'erreur $< 1/2$, tout en imposant toujours que le vérificateur ne lise qu'un nombre constant de bits de la preuve. Ceci implique la construction d'une structure algébrique complexe qui garantisse que les valeurs de toutes les petites parties de la preuve (quelques bits) dépendent de l'ensemble des bits de l'entrée, assurant ainsi que la moindre erreur dans la preuve se « propage » sur toute la preuve et soit donc détectable à la lecture de quelques bits.

Le théorème PCP donne directement un problème de maximisation qui n'admet pas de 1/2-approximation, à moins que $\mathbf{P} = \mathbf{NP}$.

Problème 29.4 (Probabilité d'acceptation maximum) Soit V un vérificateur $\mathbf{PCP}(\log n, 1)$ pour SAT. Étant donné une formule SAT ϕ, trouver une preuve qui maximise la probabilité d'acceptation de V.

Fait 29.5 *À moins que* $\mathbf{P} = \mathbf{NP}$*, il n'existe pas de* 1/2*-approximation pour le problème 29.4.*

Preuve : Si ϕ est satisfaisable, alors il existe une preuve que V accepte avec probabilité 1, et si ϕ est insatisfaisable, alors pour toute preuve, V accepte

avec probabilité $< 1/2$. Supposons qu'il existe une 1/2-approximation pour le problème 29.4. Si ϕ est satisfaisable, alors cet algorithme produit une preuve que V accepte avec une probabilité $\geqslant 1/2$. Or, on peut calculer exactement en temps polynomial la probabilité d'acceptation, simplement en exécutant V sur toutes les chaînes de bits aléatoires possibles de longueur $O(\log n)$. On peut donc, en utilisant l'algorithme d'approximation, décider SAT en temps polynomial, ce qui implique que $\mathbf{P} = \mathbf{NP}$. □

Voici un corollaire immédiat du fait 29.5.

Corollaire 29.6 *À moins que* $\mathbf{P} = \mathbf{NP}$*, il n'existe pas de PTAS pour le problème 29.4.*

Dans les sections suivantes, nous utiliserons le théorème PCP pour obtenir des résultats sur la difficulté de l'approximation pour des problèmes d'optimisation naturels ; dans chaque cas nous obtiendrons un corollaire similaire.

29.3 Difficulté de l'approximation de MAX-3SAT

MAX-3SAT est la restriction de MAX-SAT (problème 16.1) aux instances où chaque clause est composée de trois littéraux, ou moins. Ce problème joue un rôle similaire dans la difficulté de l'approximation à celui de 3SAT dans la théorie de la **NP**-difficulté : celui de problème « germe » à partir duquel on peut trouver des réductions à de nombreux autres problèmes. Voici le résultat principal de cette section :

Théorème 29.7 *Il existe une constante* $\varepsilon_M > 0$ *pour laquelle il existe une réduction écartante de SAT à MAX-3SAT qui transforme toute formule booléenne* ϕ *en une formule* ψ *telle que :*
- *si* ϕ *est satisfaisable,* $\mathrm{OPT}(\psi) = m$*, et*
- *si* ϕ *n'est pas satisfaisable,* $\mathrm{OPT}(\psi) < (1 - \varepsilon_M)m$*,*

où m *désigne le nombre de clauses de* ψ*.*

Corollaire 29.8 *À moins que* $\mathbf{P} = \mathbf{NP}$*, il n'existe pas de* $(1 - \varepsilon_M)$*-approximation pour MAX-3SAT, où* $\varepsilon_M > 0$ *est la constante définie au théorème 29.7.*

Remarquons qu'on sait qu'il est impossible de résoudre exactement (en temps polynomial) MAX-3SAT si $\mathbf{P} \neq \mathbf{NP}$. Il est intéressant d'observer que le résultat sur la difficulté de l'approximation de MAX-3SAT repose sur la même hypothèse.

Pour améliorer la lisibilité de la preuve, nous l'avons divisée en deux parties. Commençons par prouver la difficulté de l'approximation du problème suivant.

Problème 29.9 (MAX k-FONCTION SAT)[5] Soient n variables booléennes $x_1, \ldots, x_n$ et m fonctions $f_1, \ldots, f_m$, dont les valeurs ne dépendent que de k des variables booléennes. Trouver une instanciation de $x_1, \ldots, x_n$ qui maximise le nombre de fonctions satisfaites. Nous supposons ici que k est une constante fixée.
Nous obtenons ainsi une classe de problèmes pour chaque valeur de k.

Lemme 29.10 *Il existe une constante k et une réduction écartante de SAT à MAX k-FONCTION SAT, qui transforme toute formule booléenne ϕ en une instance I de MAX k-FONCTION SAT telle que :*
- *si ϕ est satisfaisable,* $\mathrm{OPT}(I) = m$, *et*
- *si ϕ est insatisfaisable,* $\mathrm{OPT}(I) < \frac{1}{2}m$,

où m est le nombre de formules dans I.

Preuve : Soit V un vérificateur de $\mathbf{PCP}(\log n, 1)$ pour SAT ; notons c et q ses paramètres. Considérons ϕ une instance de SAT de longueur n. Pour chaque chaîne de bits aléatoires r de longueur $c \log n$, V lit q bits de la preuve. Ainsi, V ne peut accéder au total qu'à moins de qn^c bits de chaque preuve de ϕ. Nous associons une variable booléenne à chacun de ces bits. Notons B l'ensemble de ces variables. La partie utile de chaque preuve correspond donc à une instanciation des variables de B.

Nous allons prouver le lemme pour $k = q$. Définissons pour chaque chaîne aléatoire r, une fonction booléenne f_r qui sera fonction de q variables de B. L'acceptation ou le rejet de V est une fonction de ϕ, r, et des q bits de la preuve lus par V. Étant donné ϕ et r, nous noterons f_r la restriction de cette fonction aux q bits de la preuve.

Clairement, il existe un algorithme polynomial qui étant donné ϕ, engendre les $m = n^c$ fonctions f_r. Si ϕ est satisfaisable, il existe une preuve que V accepte avec probabilité 1. L'instanciation correspondante des variables de B satisfait donc l'ensemble des n^c fonctions f_r. Inversement, si ϕ n'est pas satisfaisable, alors pour toute preuve, V accepte avec une probabilité $< 1/2$. Dans ce cas, toute instanciation satisfait $< \frac{1}{2}n^c$ de ces fonctions. CQFD. □

Preuve du théorème 29.7 : Le lemme 29.10 permet de transformer une formule SAT ϕ en un instance de MAX k-FONCTION SAT. Démontrons maintenant comment obtenir une formule 3SAT à partir des n^c fonctions.

Chacune des fonctions booléennes (f_r) construites au lemme 29.10 s'écrit comme une formule SAT ψ_r ayant moins de 2^q clauses. Chaque clause de ψ_r contient moins de q littéraux. Notons ψ la formule SAT définie comme la conjonctions de ces formules, c'est-à-dire $\psi = \bigwedge_r \psi_r$.

Si une instanciation satisfait la formule f_r, alors elle satisfait toutes les clauses de ψ_r. Inversement, si elle ne satisfait pas f_r, alors au moins une clause de ψ_r n'est pas satisfaite. Ainsi, si ϕ n'est pas satisfaisable, toute instanciation laisse au moins $\frac{1}{2}n^c$ clauses de ψ insatisfaites.

[5] *MAX FUNCTION SAT k-*, en anglais.

Pour conclure, il suffit maintenant de transformer ψ en une formule 3SAT. Il s'agit d'appliquer l'astuce classique qui introduit de nouvelles variables pour réduire la taille des clauses. Prenons la clause $C = (x_1 \vee x_2 \vee \cdots \vee x_k)$, avec $k > 3$. Introduisons $k-3$ nouvelles variables booléennes, $y_1, \ldots, y_{k-3}$, et posons :

$$f = (x_1 \vee x_2 \vee y_1) \wedge (\overline{y}_1 \vee x_3 \vee y_2) \wedge \cdots \wedge (\overline{y}_{k-3} \vee x_{k-1} \vee x_k).$$

Considérons une instanciation τ de $x_1, \ldots, x_k$. Si τ satisfait C, alors on peut l'étendre en une instanciation qui satisfait toutes les clauses de f. Inversement, si τ ne satisfait pas C, alors n'importe quelle instanciation de $y_1, \ldots, y_{k-3}$ laissera une clause de f (au moins) insatisfaite.

Nous appliquons cette transformation à toutes les clauses de ψ contenant plus de trois littéraux. Notons ψ' la formule 3SAT obtenue. Elle contient moins de $n^c 2^q(q-3)$ clauses. Si ϕ est satisfaisable, alors il existe une instanciation qui satisfait toutes les clauses de ψ'. Si ϕ n'est pas satisfaisable, alors toute instanciation laisse $> \frac{1}{2}n^c$ clauses insatisfaites. Il suffit donc de poser $\varepsilon_M = 1/(2^{q+1}(q-3))$ pour conclure le théorème. □

29.4 Difficulté de l'approximation de MAX-3SAT avec un nombre d'occurrences des variables borné

Pour toute constante k fixée, nous appelons MAX-3SAT(k) la restriction du problème MAX-3SAT aux formules booléennes dans lesquelles chaque variable apparaît au plus k fois. Ce problème permet d'obtenir des réductions vers des problèmes d'optimisation fondamentaux.

Théorème 29.11 *Il existe une réduction d'écart allant de MAX-3SAT vers MAX-3SAT(29) qui transforme toute formule booléenne ϕ en ψ telle que :*
- *si* $\mathrm{OPT}(\phi) = m$, *alors* $\mathrm{OPT}(\psi) = m'$, *et*
- *si* $\mathrm{OPT}(\phi) < (1-\varepsilon_M)m$, *alors* $\mathrm{OPT}(\psi) < (1-\varepsilon_b)m'$,

où m et m' sont respectivement les nombres de clauses dans ϕ et ψ, ε_M est la constante donnée par le théorème 29.7, et $\varepsilon_b = \varepsilon_M/43$.

Preuve : La preuve repose sur des graphes particuliers, les expandeurs. Rappelons (voir section 20.3) qu'un *expandeur* est un graphe $G = (V, E)$ régulier (c'est-à-dire dont tous les sommets ont le même degré), tel que pour tout sous-ensemble non vide de sommets $S \subset V$,

$$|E(S, \overline{S})| > \min(|S|, |\overline{S}|),$$

où $E(S, \overline{S})$ désigne l'ensemble des arêtes de la coupe $(S, \overline{S})$, c'est-à-dire l'ensemble des arêtes qui relient un sommet de S à un sommet de $\overline{S}$. Admettons que de tels graphes se construisent efficacement, en particulier qu'il existe un

algorithme $\mathcal{A}$ et une constante N_0 tels que pour tout $N \geqslant N_0$, $\mathcal{A}$ construit un expandeur de degré 14 à N sommets en temps polynomial en N (la remarque 29.12 éclaircit ce point).

À l'aide d'expandeurs, nous allons construire l'objet suivant qui permettra d'assurer que toute instanciation optimale est uniformément identique sur un ensemble de variables booléennes donné, c'est-à-dire soit vrai partout, soit faux partout. Soit $k \geqslant N_0$, notons G_x un expandeur de degré 14 à k sommets. Étiquetons les sommets avec des variables booléennes distinctes $x_1, \ldots, x_k$. Nous allons construire une formule sous forme normale conjonctive ψ_x sur ces variables. Pour chaque arête $x_i x_j$ de G_x, nous ajoutons deux clauses $(\overline{x}_i \vee x_j)$ et $(\overline{x}_j \vee x_i)$ à ψ_x. Une instanciation de $x_1, \ldots, x_k$ sera dite *consistante* si les variables valent toutes vrai ou toutes faux. Considérons une instanciation inconsistante. Cette instanciation partitionne les sommets de G_x en deux ensembles, S et $\overline{S}$. Quitte à échanger S et $\overline{S}$, supposons que S est le plus petit ensemble. À chaque arête de la coupe $(S, \overline{S})$, correspond une clause insatisfaite de ψ_x. Ainsi, le nombre de clauses insatisfaites $|E(S, \overline{S})|$ est supérieur à $|S| + 1$. Ceci sera crucial dans la suite.

Décrivons maintenant la réduction. Quitte à dupliquer des clauses (ce qui ne change rien d'essentiel aux propriétés d'approximabilité), nous pouvons supposer que chaque variable de l'instance SAT ϕ apparaît au moins N_0 fois.

Notons B l'ensemble des variables booléennes apparaissant dans ϕ. Nous appliquons la transformation suivante pour chaque variable x de B. Supposons que x apparaît $k \geqslant N_0$ fois dans ϕ. Associons à x un ensemble de nouvelles variables $V_x = \{x_1, \ldots, x_k\}$. Soit G_x un expandeur de degré 14 à k sommets. Nous étiquetons les sommets de G_x avec les variables de V_x et construisons la formule ψ_x décrite ci-dessus. Puis, nous remplaçons chaque occurrence de x dans ϕ par une variable distincte de V_x. Une fois cette transformation appliquée à chaque variable $x \in B$, chaque occurrence d'une variable de ϕ est remplacée par une variable distincte issue du nouvel ensemble de variables :

$$V = \bigcup_{x \in B} V_x.$$

Notons ϕ' la formule obtenue. Rappelons que nous avons également construit une formule ψ_x pour chaque variable $x \in B$.

Posons finalement :

$$\psi = \phi' \wedge \Big(\bigwedge_{x \in B} \psi_x \Big).$$

Remarquons que pour tout $x \in B$, chaque variable de V_x apparaît exactement 29 fois dans ψ — une fois dans ϕ', et 28 fois dans ψ_x. Ainsi, ψ est une instance de MAX-3SAT(29). Nous dirons que les clauses de ϕ' sont de type I, et que les autres clauses de ψ sont de type II.

Le fait important est que toute instanciation optimale de ψ satisfait toutes les clauses de type II, et doit donc être consistant sur chacun des ensembles V_x, avec $x \in B$. Par l'absurde, supposons qu'il existe une instanciation optimale τ qui ne soit pas consistante sur un ensemble V_x, pour un $x \in B$. τ partitionne les sommets de G_x en deux ensembles S et $\overline{S}$, où S est l'ensemble le plus petit. Inversons les valeurs des variables dans S, en laissant inchangées celles des autres. Il est possible que certaines clauses de type I qui étaient satisfaites par τ, ne le soient plus maintenant. Mais, chacune d'entre elles doit contenir une variable de S, et il ne peut donc y en avoir plus que $|S|$. Or, cette inversion a permis de satisfaire au moins $|S| + 1$ nouvelles clauses correspondant aux arêtes de la coupe $(S, \overline{S})$. Ainsi, l'instanciation modifiée satisfait strictement plus de clauses que τ, contradiction.

Notons m et m' les nombres de clauses de ϕ et ψ respectivement. Le nombre total d'occurrences des variables de ϕ est inférieur à $3m$. Chaque occurrence participe à 28 clauses de type II ayant deux littéraux, soit un total inférieur à $42m$ clauses de type II. Enfin, ψ contient m clauses de type I. Ainsi, $m' \leqslant 43m$.

Si ϕ est satisfaisable, alors ψ également. Enfin, supposons que $\mathrm{OPT}(\phi) < (1-\varepsilon_M)m$, c'est-à-dire que toute instanciation laisse $> \varepsilon_M m$ clauses de ϕ insatisfaites. Alors, par le fait précédent, strictement plus de $\varepsilon_M m \geqslant \varepsilon_M m'/43$ clauses de ψ sont insatisfaites. CQFD. □

Remarque 29.12 L'hypothèse que nous avons faite sur la construction efficace d'expandeurs est partiellement fausse. Nous savons seulement construire efficacement, pour tout $N \geqslant N_0$, un expandeur de taille $\leqslant N(1 + o(1))$ (voir section 29.9). Le lecteur pourra vérifier que cela suffit pour prouver le théorème 29.11.

L'exercice 29.4 généralise le théorème 29.11 en établissant la difficulté de l'approximation de MAX-3SAT(5).

29.5 Difficulté de l'approximation de la couverture par sommets et de l'arbre de Steiner

Dans cette section, nous appliquons les méthodes développées précédemment à des problèmes de graphes. Étant donné $d \geqslant 1$, notons VC(d) la restriction du problème de la couverture par sommets de taille minimum aux instances où le degré de chaque sommet est inférieur à d.

Théorème 29.13 *Il existe une réduction d'écart de MAX-3SAT(29) à VC(30) qui transforme toute formule booléenne ϕ en un graphe $G = (V, E)$ tel que :*

- *si* $\mathrm{OPT}(\phi) = m$, *alors* $\mathrm{OPT}(G) \leqslant \frac{2}{3}|V|$, *et*
- *si* $\mathrm{OPT}(\phi) < (1 - \varepsilon_b)m$, *alors* $\mathrm{OPT}(G) > (1 + \varepsilon_v)\frac{2}{3}|V|$,

où m est le nombre de clauses de ϕ, ε_b est la constante du théorème 29.11, et $\varepsilon_v = \varepsilon_b/2$.

Preuve : Quitte à répéter des littéraux dans certaines clauses, supposons que toutes les clauses de ϕ ont exactement trois littéraux. Nous utilisons la transformation classique qui associe trois sommets de G à chaque clause de ϕ. Chacun de ces sommets est étiqueté par un littéral de la clause. Ainsi, $|V| = 3m$. Les arêtes de G sont de deux types (voir illustration ci-dessous) :

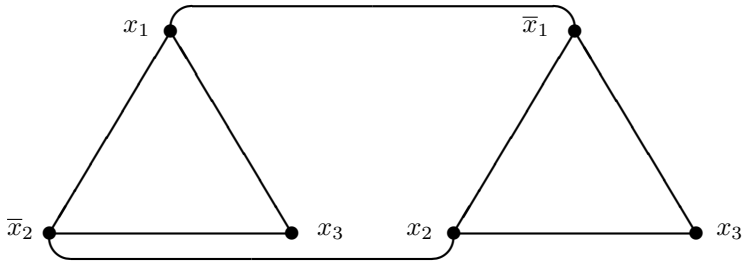
- G a trois arêtes reliant les trois sommets correspondant à chaque clause de ϕ, et
- G a une arête uv pour chaque paire de sommets $u, v \in V$ dont les littéraux associés sont les négations l'un de l'autre.

Chaque sommet de G a deux arêtes du premier type et moins de 28 du second. Le degré de G est donc inférieur à 30.

Nous affirmons que la taille d'un stable maximum de G est exactement $\mathrm{OPT}(\phi)$. Considérons une instanciation optimale de ϕ et sélectionnons pour chaque clause un sommet correspondant au littéral satisfait. L'ensemble des sommets sélectionnés forme clairement un stable de G. Inversement, prenons un stable I de G, et donnons la valeur vraie aux littéraux correspondants. N'importe quelle extension de cette instanciation satisfera au moins $|I|$ clauses.

Or, le complémentaire de tout stable maximum de G est une couverture par sommets minimum de G. Ainsi, si $\mathrm{OPT}(\phi) = m$ alors $\mathrm{OPT}(G) = 2m$. Et si $\mathrm{OPT}(\phi) < (1 - \varepsilon_b)m$, alors $\mathrm{OPT}(G) > (2 + \varepsilon_b)m$. CQFD. □

À titre d'illustration, considérons la formule $(x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$. En voici le graphe associé par la réduction du théorème 29.13 :



Théorème 29.14 *Il existe une réduction d'écart de VC(30) au problème de l'arbre de Steiner. Elle transforme toute instance $G = (V, E)$ de VC(30) en une instance $H = (R, S, \text{coût})$ de l'arbre de Steiner, où R et S sont respectivement les sommets requis et Steiner de H et* coût *est une métrique sur $R \cup S$, telle que :*

- *si* $\mathrm{OPT}(G) \leqslant \frac{2}{3}|V|$, *alors* $\mathrm{OPT}(H) \leqslant |R| + \frac{2}{3}|S| - 1$, *et*

– *si* $\mathrm{OPT}(G) > (1+\varepsilon_v)\frac{2}{3}|V|$, *alors* $\mathrm{OPT}(H) > (1+\varepsilon_s)(|R|+\frac{2}{3}|S|-1)$, *où* $\varepsilon_s = 4\varepsilon_v/97$ *et* ε_v *est la constante du théorème 29.13.*

Preuve : Nous allons construire un graphe $H = (R, S, \text{coût})$ tel que G admet une couverture par sommets de taille c ssi H admet un arbre de Steiner de coût $|R| + c - 1$. Pour chaque arête $e \in E$, nous créons un sommet requis r_e dans H, et pour chaque sommet $v \in V$, un sommet Steiner s_v. Les coûts sont définis ainsi : une arête ayant pour extrémités deux sommets Steiner coûte 1, et une arête ayant pour extrémités deux sommets requis coûte 2 ; une arête $r_e s_v$ coûte 1 si l'arête e est incidente au sommet v dans G et 2 sinon.

Démontrons que G admet une couverture par sommets de taille c ssi H admet un arbre de Steiner de coût $|R| + c - 1$. Pour le sens direct, notons S_c l'ensemble des sommets Steiner de H qui correspondent aux sommets de la couverture. Observons qu'il existe un arbre dans H qui couvre $R \cup S_c$ en n'utilisant que des arêtes de coût 1 (car toute arête $e \in E$ est incidente à un sommet de la couverture). C'est un arbre de Steiner de coût $|R| + c - 1$.

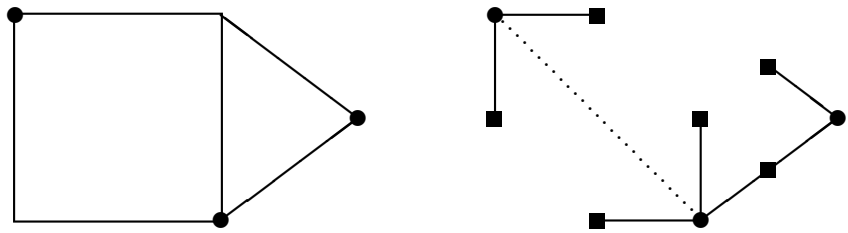
Pour la réciproque, soit T un arbre de Steiner de H coûtant $|R| + c - 1$. Nous allons démontrer qu'on peut transformer T en un arbre de Steiner de même coût, mais n'utilisant que des arêtes de coût 1. Ce nouvel arbre contiendra alors exactement c sommets Steiner ; de plus, tout sommet requis de H devra être relié par une arête de coût unitaire à un de ces sommets Steiner. Les c sommets correspondants dans G formeront donc bien une couverture de G.

Soit uv une arête de coût 2 de T. Si u est un sommet Steiner, nous pouvons supprimer l'arête uv, ce qui coupe T en deux composantes, qu'on recolle en ajoutant une arête reliant v à l'un des sommets requis de l'autre composante. Cette opération ne modifie pas le coût de T. Supposons donc à présent que u et v sont tous deux requis. Notons e_u et e_v les arêtes qui correspondent à ces deux sommets dans G. Comme G est connexe, il existe un chemin p reliant l'une des extrémités de e_u à une extrémité de e_v dans G. Ôtons donc l'arête uv de T, cela divise T en deux composantes. Notons R_1 et R_2 les ensembles des sommets requis de chacune de ces composantes. Clairement, u et v n'appartiennent pas à la même composante, le chemin p doit donc contenir deux arêtes adjacentes ab et bc, telles que leurs sommets correspondants w et w' dans H appartiennent respectivement à R_1 et R_2. Soit s_b le sommet Steiner de H correspondant à b. Rajoutons maintenant les arêtes $s_b w$ et $s_b w'$, ceci reconnecte les deux composantes. Remarquons que ces deux arêtes ont un coût unitaire.

Finalement, si $\mathrm{OPT}(G) \leqslant \frac{2}{3}|V|$, alors $\mathrm{OPT}(H) \leqslant |R| + \frac{2}{3}|S| - 1$, et si $\mathrm{OPT}(G) > (1+\varepsilon_v)\frac{2}{3}|V|$, alors $\mathrm{OPT}(H) > |R| + (1+\varepsilon_v)\frac{2}{3}|S| - 1$. CQFD. □

Cette réduction est illustrée ci-dessous. Le graphe G est une instance du problème de la couverture par sommets. Les sommets marqués par des ronds forment une couverture. L'arbre de Steiner correspondant est représenté dans le graphe H. Les sommets requis sont marqués d'un carré noir et les trois sommets Steiner par des ronds (les autres sommets Steiner ne sont pas représentés

par souci de clarté). L'arête entre les deux sommets Steiner est dessinée en pointillés pour la distinguer des autres, qui connectent un sommet Steiner à un sommet requis.



G *H*

29.6 Difficulté de l'approximation de Clique

Les meilleurs algorithmes d'approximation connus pour certains problèmes, dont Clique, sont extrêmement peu performants — au sens que la solution produite est à peine meilleure celle qui consiste à sélectionner une solution réalisable triviale. Des résultats précieux sur la difficulté de l'approximation de ces problèmes ont permis récemment d'expliquer pourquoi : ces problèmes sont en fait (intrinsèquement) inapproximables. Cette section démontre ce résultat pour Clique :

Problème 29.15 (Clique) Étant donné un graphe non orienté $G = (V, E)$ muni de poids positifs sur les sommets, trouver une clique de poids maximum. Une *clique* de G est un sous-ensemble de sommets, $S \subseteq V$, tel que toute paire de sommets $u, v \in S$ est connectée, c'est-à-dire $uv \in E$. Son *poids* est la somme des poids de ses sommets.

Considérons la version « cardinalité » de ce problème, c'est-à-dire le cas où tous les sommets ont pour poids 1. Dans cette section, nous allons démontrer qu'il existe une constante $\varepsilon_q > 0$, telle qu'il n'existe aucune $(1/n^{\varepsilon_q})$-approximation pour ce problème, à moins que $\mathbf{P} \neq \mathbf{NP}$. Commençons par prouver le résultat plus faible suivant.

Lemme 29.16 *Il existe deux constantes fixées b et q, et une réduction d'écart de SAT à Clique qui transforme toute formule booléenne ϕ de taille n en un graphe $G = (V, E)$, avec $|V| = 2^q n^b$, tel que :*
- *si ϕ est satisfaisable,* $\mathrm{OPT}(G) \geqslant n^b$, *et*
- *si ϕ n'est pas satisfaisable,* $\mathrm{OPT}(G) < \frac{1}{2} n^b$.

Preuve : Soit F un vérificateur $\mathbf{PCP}(\log n, 1)$ pour SAT qui utilise $b \log n$ bits aléatoires et lit q bits de la preuve. Transformons toute instance ϕ de SAT en un graphe $G = (V, E)$ de taille n, comme suit. G a un sommet $v_{r,\tau}$

pour chaque paire constituée d'une chaîne de bits aléatoires r de longueur $b \log n$ et d'une instanciation τ des q variables booléennes. Ainsi, $|V| = 2^q n^b$.

Notons $Q(r)$ les q positions de la preuve lues par F à la donnée de la chaîne de bits aléatoires r. Nous dirons qu'un sommet $v_{r,\tau}$ est *acceptant* si F accepte quand r et τ sont respectivement la chaîne de bits aléatoires et les valeurs lues dans les $Q(r)$ positions de la preuve ; sinon, $v_{r,\tau}$ est dit *rejetant*. Nous dirons que deux sommets v_{r_1,τ_1} et v_{r_2,τ_2} sont *consistants* si τ_1 et τ_2 coïncident sur toutes les positions de la preuve, communes à $Q(r_1)$ et $Q(r_2)$. Clairement, lorsque deux sommets distincts sont consistants, nous avons $r_1 \neq r_2$. Deux sommets distincts v_{r_1,τ_1} et v_{r_2,τ_2} sont reliés par une arête dans G ssi ils sont consistants et tous deux acceptants. Un sommet $v_{r,\tau}$ est dit *consistant* avec une preuve p si les valeurs des positions $Q(r)$ de p sont données par τ.

Si ϕ est satisfaisable, il existe une preuve p que F accepte quelle que soit la chaîne de bits aléatoire r. Pour chaque r, notons $p(r)$ les valeurs des bits de p sur les positions $Q(r)$. L'ensemble des sommets $\{v_{r,p(r)} : |r| = b \log n\}$ forme donc une clique de taille n^b dans G.

Supposons maintenant que ϕ est insatisfaisable. Soit C une clique de G. Comme les sommets de C sont consistants deux à deux, il existe une preuve p qui est consistante avec tous les sommets de C. Ainsi, la probabilité d'acceptation de F sur la preuve p est supérieure à $|C|/n^b$ (car les sommets de C correspondent tous à une chaîne aléatoire différente). Puisque la probabilité d'acceptation de toute preuve est $< 1/2$, la taille maximum d'une clique de G est $< \frac{1}{2} n^b$. □

Une conséquence du lemme 29.16 est qu'il n'existe pas de 1/2-approximation pour Clique, sauf si $\mathbf{P} = \mathbf{NP}$. Remarquons que le facteur d'approximation limite obtenu est exactement la borne sur la probabilité d'erreur d'une preuve vérifiable stochastiquement de SAT. En répétant classiquement un nombre constant de fois la vérification, nous pouvons réduire cette constante à $1/k$ pour n'importe quelle constante k, et obtenir un résultat similaire sur Clique. Pour établir le résultat promis, nous devons rendre la probabilité d'erreur inférieure à un polynôme en n. Pour cela, nous allons introduire une généralisation de la classe **PCP**. Nous définissons deux paramètres supplémentaires c et s, appelés respectivement *complétude* et *adéquation*.[6] Un langage L appartient à $\mathbf{PCP}_{c,s}[r(n), q(n)]$ s'il existe un vérificateur V, qui sur toute entrée x de taille n, utilise $O(r(n))$ bits aléatoires, lit $O(q(n))$ bits de la preuve, et satisfait les conditions suivantes :

– si $x \in L$, il existe une preuve y que V accepte avec probabilité $\geqslant c$,

– si $x \notin L$, alors pour toute preuve y, V accepte y avec probabilité $< s$.

La classe définie précédemment est $\mathbf{PCP}[r(n), q(n)] = \mathbf{PCP}_{1,\frac{1}{2}}[r(n), q(n)]$. En toute généralité, c et s sont des fonctions de n.

Nous aimerions obtenir une caractérisation de **NP** sous forme d'une classe PCP d'adéquation inversement polynomiale. Une façon évidente de réduire l'adéquation est de répéter l'exécution d'un vérificateur $\mathbf{PCP}[\log n, 1]$ un

[6] *Completeness* et *soundness*, en anglais.

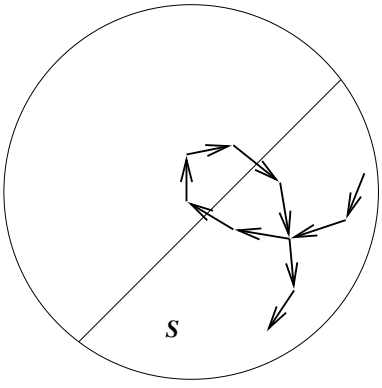
certain nombre de fois et d'accepter s si le vérificateur a accepté chaque exécution. Exécuter k fois le vérificateur ramène l'adéquation à $1/2^k$; néanmoins, ceci augmente considérablement le nombre de bits utilisés, $O(k \log n)$, et le nombre de bits lus dans la preuve, $O(k)$. Remarquons que le nombre de sommets dans le graphe construit au lemme 29.16 est $2^{O(r(n)+q(n))}$. Pour obtenir ainsi une adéquation inversement polynomiale, nous aurions besoin que $k = \Omega(\log n)$, et pour cette valeur de k, le nombre de bits lus dans la preuve serait $O(\log n)$, ce qui n'est pas un problème, mais le nombre de bits aléatoires utilisés serait $O(\log^2 n)$, ce qui conduirait à un graphe de taille superpolynomiale.

Une idée astucieuse permet de contourner cette difficulté. Nous allons utiliser un expandeur de degré constant pour générer $O(\log n)$ chaînes de $b \log n$ bits chacune, en utilisant uniquement $O(\log n)$ bits réellement aléatoires. Le vérificateur sera exécuté avec ces $O(\log n)$ chaînes pour chaînes aléatoires. Clairement, ce ne sont pas des chaînes réellement aléatoires, mais les propriétés des expandeurs font qu'elles sont « presque aléatoires » — la probabilité d'erreur chute encore exponentiellement avec le nombre d'exécutions successives du vérificateur.

Soit H un expandeur de degré constant à n^b sommets, où chaque sommet est étiqueté par une unique chaîne de $b \log n$ bits. Une marche aléatoire sur H de longueur $O(\log n)$ est encodée par $O(\log n)$ bits seulement, $b \log n$ bits pour sélectionner le sommet de départ uniformément, et un nombre constant de bits pour sélectionner le sommet suivant à chaque fois — remarquons que la marche aléatoire est initialisée selon la distribution stationnaire, qui est uniforme car le graphe est régulier. Voici la propriété exacte sur les expandeurs dont nous aurons besoin.

Théorème 29.17 *Soit S un sous-ensemble de sommets de H de taille $< (n^b)/2$. Il existe une constante k telle que :*

$$\mathbf{Pr}[\textit{ tous les sommets d'une marche aléatoire de longueur } k \log n \textit{ soient dans } S\,] < \tfrac{1}{n}.$$



Une justification intuitive du théorème 29.17, est qu'une fraction constante des arêtes incidentes à un sommet de S ont leur autre extrémité dans $\overline{S}$. Ceci permet aux marches aléatoires de s'échapper rapidement de S. La figure précédente illustre une marche aléatoire dans H qui ne reste pas dans S.

Théorème 29.18 $\mathbf{NP} = \mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n]$

Preuve : Démontrons l'inclusion difficile :

$$\mathbf{PCP}_{1,\frac{1}{2}}[\log n, 1] \subseteq \mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n].$$

L'autre inclusion est l'objet de l'exercice 29.5. Soit $L \in \mathbf{PCP}_{1,\frac{1}{2}}[\log n, 1]$. Soit F un vérificateur de L qui utilise $b \log n$ bits aléatoires et lit q bits de la preuve, où b et q sont des constantes.

Construisons un vérificateur $\mathbf{PCP}_{1,\frac{1}{2}}[\log n, \log n]$, F', pour L. F' commence par construire l'expandeur H défini ci-dessus. Puis, il génère une marche aléatoire dans H de longueur $k \log n$, en utilisant $O(\log n)$ bits aléatoires. Ceci est accompli en un temps polynomial. Les étiquettes des sommets de la marche définissent une suite de $k \log n + 1$ chaînes de $b \log n$ bits. Ces $k \log n + 1$ chaînes sont utilisées comme chaînes « aléatoires » pour exécuter autant de fois le vérificateur F. Enfin, F' accepte ssi F accepte toutes les $k \log n + 1$ exécutions.

Considérons un mot $x \in L$, et une preuve p de x que F accepte avec probabilité 1. Clairement, sur la donnée de p, F' accepte x avec probabilité 1. Ainsi la complétude du nouveau système de preuve est 1.

Considérons maintenant un mot $x \notin L$, et une preuve p quelconque soumise à F'. Le vérificateur F accepte p pour moins de $(n^b)/2$ chaînes aléatoires de longueur $b \log n$. Soit S l'ensemble des sommets correspondants dans H, $|S| < (n^b)/2$. Maintenant, F' accepte x et p ssi la marche aléatoire reste intégralement dans S. Comme la probabilité de cet événement est $< 1/n$, l'adéquation de F' est $1/n$. Concluons en remarquant que F' n'utilise que $O(\log n)$ bits aléatoires et lit $O(\log n)$ bits de la preuve. □

Théorème 29.19 *Pour des constantes fixées b et q, il existe une réduction d'écart de SAT à Clique qui transforme toute formule booléenne ϕ de taille n en un graphe $G = (V, E)$, avec $|V| = n^{b+q}$, tel que :*
- *si ϕ est satisfaisable,* $\mathrm{OPT}(G) \geqslant n^b$, *et*
- *si ϕ n'est pas satisfaisable,* $\mathrm{OPT}(G) < n^{b-1}$.

Preuve : Soit F un vérificateur $\mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n]$ pour SAT qui utilise $b \log n$ bits aléatoires et lit $q \log n$ bits de la preuve. Nous appliquons exactement la transformation d'une instance de SAT en un graphe G, définie au lemme 29.16. La seule différence est que le nombre de sommets est plus grand car le nombre de bits lus dans la preuve a augmenté.

La preuve que la construction est correcte, est identique à celle du lemme 29.16. Si ϕ est satisfaisable, considérons une bonne preuve p, et sélectionnons les n^b sommets de G qui sont consistants avec p, un pour chaque choix de la chaîne aléatoire. Ces sommets forment une clique de G. Inversement, toute clique C de G engendre une preuve qui est acceptée par F avec probabilité $\geqslant |C|/n^b$. Puisque l'adéquation de F est $1/n$, si ϕ n'est pas satisfaisable, la taille maximum d'une clique de G est $< n^{b-1}$. □

Corollaire 29.20 *Il n'existe pas de* $1/(n^{\varepsilon_q})$*-approximation pour le problème de la clique de taille maximum, à moins que* $\mathbf{P} = \mathbf{NP}$*, où* $\varepsilon_q = 1/(b+q)$ *pour les constantes* b *et* q *définies au théorème 29.19.*

29.7 Difficulté de l'approximation de la couverture par ensembles

Nous l'avons dit au chapitre 2, l'algorithme glouton pour la couverture par ensembles, qui est probablement la première idée qui vienne à l'esprit, est toujours pour l'essentiel le meilleur algorithme connu. Pourtant, le problème de la couverture par ensembles est sans doute le problème le plus important en algorithmique d'approximation, et beaucoup d'efforts ont été dépensés pour trouver un algorithme plus performant.

Nous présentons ici le résultat remarquable démontrant que le facteur d'approximation de l'algorithme glouton est optimal à une constante multiplicative près. Il a été démontré par la suite que ce facteur est optimal à des termes d'ordre inférieur près (voir section 29.9). Ce résultat devrait donc clore la question de l'approximabilité de ce problème central.

29.7.1 Caractérisation de NP par deux prouveurs en une ronde

Remarquons que pour démontrer la difficulté de l'approximation de MAX-3SAT et Clique (théorèmes 29.7 et 29.19), nous n'avions pas besoin de connaître exactement quel était le type de requêtes formulées par le vérificateur sur la preuve — nous avions uniquement utilisé une borne sur le nombre de requêtes (c'est-à-dire sur le nombre de bits lus). Nous aurons besoin ici d'une description plus précise de la formulation des requêtes ; en particulier, nous allons démontrer qu'un type particulier de vérificateur suffit. Nous introduisons dans ce but un nouveau modèle de système de preuve vérifiable stochastiquement, les *systèmes à deux prouveurs en une ronde.*[7] On comprend mieux ce modèle en l'interprétant comme un jeu entre prouveurs et vérificateur. Chaque prouveur essaye de tricher – il essaye de convaincre le vérificateur qu'une instance « négative » du langage L est bien dans L. La

[7] *Two-prover one-round proof system*, en anglais.

question est : existe-t-il un vérificateur dont la probabilité d'être induit en erreur par le(s) prouveur(s) est $< 1/2$ pour toute instance « négative » ?

Dans le modèle à deux prouveurs, le vérificateur peut questionner les deux prouveurs, P_1 et P_2, qui ne communiquent pas entre eux. Comme le vérificateur peut croiser les réponses des prouveurs, cela limite la puissance de ces prouveurs. Nous restreignons également le vérificateur pour obtenir une nouvelle caractérisation de **NP**. Dans le modèle à une ronde, le vérificateur n'est autorisé qu'à une ronde de communication avec chaque prouveur. La formulation la plus simple est la suivante. Nous supposons que les deux preuves sont écrites sur deux alphabets Σ_1 et Σ_2. En général, les tailles de ces alphabets ne sont pas bornées et peuvent dépendre de la taille de l'entrée. Le vérificateur est autorisé à lire une unique position de chacune des deux preuves.

Le modèle à deux prouveurs en une ronde est caractérisé par trois paramètres : la complétude, l'adéquation, et le nombre de bits aléatoires utilisés par le vérificateur, notés respectivement c, s et $r(n)$. Cela définit la classe $\mathbf{2P1R}_{c,s}(r(n))$. Un langage L appartient à $\mathbf{2P1R}_{c,s}(r(n))$ s'il existe un vérificateur en temps polynomial qui utilise $O(r(n))$ bits (parfaitement) aléatoires, et tel que :

- pour toute entrée $x \in L$, il existe une paire de preuves $y_1 \in \Sigma_1^*$ et $y_2 \in \Sigma_2^*$ que V accepte avec probabilité $\geqslant c$,
- pour toute entrée $x \notin L$ et toute paire de preuves $y_1 \in \Sigma_1^*$ et $y_2 \in \Sigma_2^*$, V accepte avec probabilité $< s$.

Le théorème PCP implique (et est en fait équivalent à) l'existence d'une réduction écartante de SAT à MAX-3SAT(5) (voir théorème 29.7 et exercices 29.3 et 29.4). Nous allons utiliser ce résultat pour démontrer :

Théorème 29.21 *Il existe une constante $\varepsilon_P > 0$ telle que :*
$\mathbf{NP} = \mathbf{2P1R}_{1,1-\varepsilon_P}(\log(n))$.

Preuve : Prouvons l'inclusion difficile, soit $\mathbf{NP} \subseteq \mathbf{2P1R}_{1,1-\varepsilon_P}(\log(n))$, l'inclusion inverse est l'objet de l'exercice 29.7. De toute façon, c'est cette inclusion qui impliquera que $\mathrm{SAT} \in \mathbf{2P1R}_{1,1-\varepsilon_P}(\log(n))$.

D'après le théorème 29.7 et l'exercice 29.4, il existe une réduction écartante de SAT à MAX-3SAT(5)[8]. Plus précisément, il existe une constante $\varepsilon_5 > 0$ et une réduction Γ de SAT à MAX-3SAT(5) qui transforme toute formule booléenne ϕ en ψ telle que :

- si ϕ est satisfaisable, $\mathrm{OPT}(\psi) = m$, et
- si ϕ n'est pas satisfaisable, $\mathrm{OPT}(\psi) < (1 - \varepsilon_5)m$,

où m est le nombre de clauses de ψ.

Le vérificateur V à deux prouveurs en une ronde pour SAT fonctionne comme suit. Étant donné une formule SAT ϕ, il utilise la réduction évoquée ci-

[8] Le fait que le nombre d'occurrences soit borné dans MAX-3SAT n'est pas essentiel pour prouver ce théorème ; mais nous en aurons besoin pour la réduction principale.

dessus pour construire l'instance ψ de MAX-3SAT(5). Le vérificateur s'attend à ce que P_1 contienne une instanciation optimale τ pour ψ et que P_2 contienne pour chaque clause l'instanciation des trois variables dans τ (ainsi, $|\Sigma_1| = 2$ et $|\Sigma_2| = 2^3$). Il utilise $O(\log n)$ bits aléatoires pour tirer une clause aléatoire C de ψ, et une variable booléenne aléatoire x apparaissant dans C. V obtient la valeur de x et celles des trois variables de C en interrogeant P_1 et P_2 respectivement. Il accepte ssi C est satisfaite et les deux preuves attribuent la même valeur à x.

Si ϕ est satisfaisable, alors ψ aussi, et il existe deux preuves y_1 et y_2 telles que V accepte avec probabilité 1.

Supposons à présent que ϕ n'est pas satisfaisable. Toute instanciation de ψ laisse une fraction $> \varepsilon_5$ des clauses insatisfaites. Considérons une paire de preuves (y_1, y_2). Voyons y_1 comme une instanciation τ. La probabilité que la clause aléatoire C sélectionnée par V ne soit pas satisfaite est $> \varepsilon_5$. Dans ce cas, si l'instanciation de C contenue dans y_2 satisfait C, alors y_1 et y_2 ne peuvent pas être consistants, et le vérificateur le détecte avec probabilité $\geqslant 1/3$. Par conséquent, V rejette avec probabilité $> \varepsilon_5/3$. □

Remarque 29.22 En utilisant des techniques classiques (voir exercice 29.8), on peut modifier Γ pour que l'instance de MAX-3SAT(5) produite vérifie les *conditions uniformes* suivantes : chaque variable booléenne apparaît exactement dans 5 clauses, et chaque clause contient exactement 3 variables distinctes (niées ou non). Cette modification change la valeur de la constante ε_5 en $\varepsilon'_5 > 0$. Nous utiliserons ces conditions uniformes dans la réduction principale.

Remarque 29.23 Sous les conditions uniformes, si ψ a n variables, alors il contient $5n/3$ clauses. Ainsi, les longueurs des deux preuves sont respectivement n et $5n/3$. Pour construire la réduction principale, nous aurons besoin que les preuves aient la même longueur. Pour cela, il suffit de répéter la première preuve 5 fois et la seconde 3 fois. Le vérificateur interrogera une copie aléatoire de chaque preuve. On vérifie facilement que le théorème 29.21 est encore vrai (même si les « copies » sont différentes).

29.7.2 Gadget

Le système d'ensembles suivant est le gadget clé de la réduction :

$$(U, C_1, \ldots, C_m, \overline{C}_1, \ldots, \overline{C}_m),$$

où U est l'ensemble univers et $C_1, \ldots, C_m$ sont des sous-ensembles de U. Clairement, U est couvert en sélectionnant un ensemble C_i et son complémentaire $\overline{C}_i$. Une telle couverture sera dite *bonne*. Une couverture qui ne contient pas un ensemble et son complémentaire sera dite *mauvaise*. Le théorème suivant, qu'on démontre par la méthode probabiliste (voir exercice 29.9), affirme l'existence de systèmes de ce type dont les tailles des couvertures bonnes et des

couvertures mauvaises sont très différentes. De plus, on peut construire de tels systèmes efficacement et avec forte probabilité.

Théorème 29.24 *Il existe un polynôme à deux variables $p(.,.)$ et un algorithme randomisé qui génère pour tous m et l, un système d'ensembles*

$$(U, C_1, \ldots, C_m, \overline{C}_1, \ldots, \overline{C}_m),$$

avec $|U| = p(m, 2^l)$, tel qu'avec probabilité $> 1/2$, la taille de toute couverture mauvaise est $> l$. De plus, le temps d'exécution de l'algorithme est polynomial en $|U|$.

De manière informelle, une bonne couverture est très cohérente — il s'agit de sélectionner un ensemble C_i et son complémentaire. L'acceptation d'un système de preuves à deux prouveurs en une ronde implique aussi de la coordination — pour chaque chaîne aléatoire r, le vérificateur interroge les deux preuves et accepte si les réponses sont cohérentes. Nous espérons que cette remarque éclairera le lecteur sur la pertinence du choix de ce système de preuve pour caractériser la difficulté de l'approximation de la couverture par sommets.

29.7.3 Réduire la probabilité d'erreur par répétitions parallèles

Avant de présenter la réduction, nous souhaitons améliorer l'adéquation du système de preuve à deux prouveurs en une ronde pour SAT. Classiquement, on utilise la répétition parallèle. Le vérificateur sélectionne k clauses aléatoires indépendamment, et une variable booléenne aléatoire dans chacune des clauses. Puis il interroge P_1 pour obtenir les valeurs des k variables et P_2 pour les k clauses, et accepte ssi toutes les réponses sont acceptantes. On est en droit d'espérer que la probabilité que les prouveurs réussissent à tricher chute à $< (1 - \varepsilon_P)^k$.

De façon surprenante, ce n'est pas le cas. Comme chacun des prouveurs est autorisé à regarder l'ensemble des k requêtes avant de donner ses k réponses (il n'y a qu'une ronde), il peut réussir à accorder ses réponses et donc à tricher avec une probabilité supérieure. L'exemple 29.25 illustre ce fait dans un cadre plus simple. Si on obligeait les prouveurs à donner la réponse à chaque question avant de lire la suivante, la probabilité d'erreur chuterait exponentiellement comme on peut s'y attendre ; cependant, cela nécessite k rondes de communication et donc sort du modèle à deux prouveurs en une ronde.

Exemple 29.25 Considérons le problème suivant où deux prouveurs qui ne communiquent pas entre eux, cherchent à tomber d'accord sur un bit aléatoire. Le vérificateur donne deux bits aléatoires indépendants r_1 et r_2 à P_1 et P_2, respectivement. Le protocole réussit si les deux prouveurs s'accordent sur l'un des deux bits, c'est-à-dire soit les deux prouveurs renvoient

$(1, r_1)$, soit les deux prouveurs renvoient $(2, r_2)$; le premier élément de la paire indique lequel des bits les prouveurs renvoient, et le second donne la valeur du bit lui-même. Comme P_1 ne connaît pas r_2 et P_2 ne connaît pas r_1, la probabilité de succès est 1/2.

Considérons maintenant des répétitions parallèles de ce protocole. Le vérificateur donne deux bits, r_1 et s_1, à P_1 et deux bits, r_2 et s_2, à P_2. Les quatre bits sont aléatoires et indépendants. Les prouveurs réussissent ssi ils s'accordent sur l'un des r_i et l'un des s_i.

On pourrait s'attendre à ce que la probabilité de succès soit 1/4. Cependant, en coordonnant astucieusement les réponses, les prouveurs peuvent atteindre 1/2 comme suit. Les réponses de P_1 sont $(1, r_1)$ et $(2, r_1)$, et celle de P_2 sont $(1, s_2)$ et $(2, s_2)$. Les prouveurs réussissent ssi $r_1 = s_2$, ce qui arrive avec probabilité 1/2. □

Malgré cette difficulté, il est toujours possible de prouver que la probabilité d'erreur décroît exponentiellement avec k. Cependant, la preuve est loin d'être simple.

Théorème 29.26 *Soit $\delta < 1$ la probabilité d'erreur d'un système de preuve à deux prouveurs en une ronde. Alors, la probabilité d'erreur de k répétitions parallèles est inférieure à δ^{dk}, où d est une constante qui ne dépend que de la longueur des réponses dans le système de preuve initial.*

29.7.4 La réduction

Voici le résultat principal.

Théorème 29.27 *Il existe une constante $c > 0$ et une réduction écartante randomisée Γ de SAT au problème de la couverture par ensembles de taille minimum, de temps d'exécution $n^{O(\log \log n)}$, qui transforme toute formule booléenne ϕ en un système d'ensembles $\mathcal{S}$ sur un ensemble univers de taille $n^{O(\log \log n)}$ tel que :*

- *si ϕ est satisfaisable, $\mathrm{OPT}(\mathcal{S}) = 2n^k$, et*
- *si ϕ n'est pas satisfaisable, $\mathbf{Pr}[\mathrm{OPT}(\mathcal{S}) > cn^k k \log n] > 1/2$,*

où le paramètre k vaut $O(\log \log n)$ et n est la longueur (commune) de chacune des deux preuves de SAT dans le modèle à deux prouveurs en une ronde (voir remarque 29.23) ; n est un polynôme en la taille de ϕ.

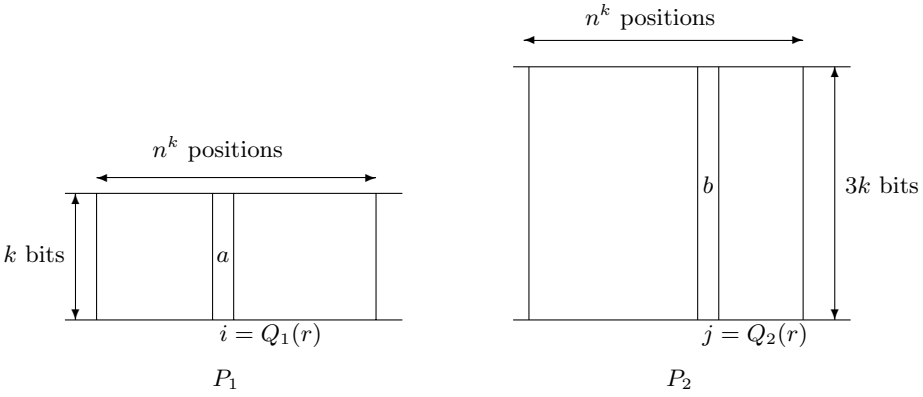
Remarque 29.28 Nous nous écartons ici légèrement des définitions, car normalement le temps d'exécution d'une réduction écartante est polynomial.

Preuve : Soit V le vérificateur à deux prouveurs en une ronde pour SAT, décrit au théorème 29.21. Imposons de plus sans perte de généralité que la formule MAX-3SAT(5) produite par V vérifie les conditions uniformes de la remarque 29.22 et que les deux preuves interrogées par V ont la même longueur n (voir remarque 29.23). Notons ψ la formule MAX-3SAT(5) engendrée par V à partir de la formule SAT ϕ.

Soit V' le vérificateur à deux prouveurs en une ronde qui exécute k répétitions parallèles de V, tel que décrit à la section 29.7.3. La longueur de chaque preuve est maintenant n^k. Chaque position de P_1 contient les valeurs de k variables booléennes (pas nécessairement distinctes) et chaque position de P_2 contient une instanciation des $3k$ variables booléennes apparaissant dans k clauses. Ainsi, les preuves P_1 et P_2 sont écrites sur les alphabets Σ_1 et Σ_2 de tailles respectives 2^k et 2^{3k}. Nous fixons k à $O(\log \log n)$ pour des raisons qui apparaîtront plus tard.

Le vérificateur V' utilise les bits aléatoires pour sélectionner k clauses aléatoires de ψ, et une variable booléenne dans chacune dans ces k clauses, c'est-à-dire une position dans P_1 et une position dans P_2. Il s'agit donc de sélectionner un élément parmi n^k puis un parmi 3^k, respectivement. Le nombre total de chaînes de bits aléatoires différentes nécessaires est donc $(3n)^k$. Notons $Q_1(r)$ et $Q_2(r)$ les positions lues dans P_1 et P_2 respectivement, quand la chaîne aléatoire est r.

Supposons que les réponses des positions $Q_1(r)$ et $Q_2(r)$ sont a et b, respectivement. Rappelons que V' accepte sur la chaîne aléatoire r ssi b satisfait l'ensemble des k clauses sélectionnées ; de plus, a et b assignent les mêmes valeurs aux k variables choisies. Une fois connues r et la réponse b à $Q_2(r)$, l'unique réponse acceptable à $Q_1(r)$ est complètement spécifiée. Notons $\pi(r, b)$ cette réponse (c'est une *projection* de b).



Les paramètres m et l du gadget sont fixés comme suit. Posons $m = |\Sigma_1| = 2^k$, et $l = O(k \log n) = O(\log n \log \log n)$. Soit $(U, C_1, \ldots, C_{2^k}, \overline{C}_1, \ldots, \overline{C}_{2^k})$ le gadget correspondant à ces paramètres. Nous pouvons associer à chaque réponse $a \in \Sigma_1$, un ensemble unique C_a. D'après le théorème 29.24, $|U| = p(2^k, 2^l) = n^{O(\log \log n)}$, et on peut construire ce gadget stochastiquement en temps polynomial en $|U|$.

Le gadget est construit une fois pour toute, et, d'après le théorème 29.24, a les valeurs souhaitées des paramètres avec probabilité $> 1/2$. Nous supposerons désormais que c'est le cas. Nous faisons $(3n)^k$ copies du gadget sur des ensembles univers disjoints. Nous associons à chaque chaîne aléatoire r une copie distincte du gadget, notée $(U^r, C_1^r, \ldots, C_{2^k}^r, \overline{C}_1^r, \ldots, \overline{C}_{2^k}^r)$.

La réduction Γ transforme ϕ en l'instance suivante de la couverture par ensembles. L'ensemble univers est

$$\mathcal{U} = \bigcup_r U^r,$$

où l'union est prise sur l'ensemble des $(3n)^k$ chaînes aléatoires. Clairement, $|\mathcal{U}| = |U|(3n)^k = n^{O(\log\log n)}$. Les sous-ensembles de $\mathcal{U}$ dans $\mathcal{S}$ sont de deux types. Nous associons à chaque position i de P_1 et réponse $a \in \Sigma_1$, l'ensemble

$$S_{i,a} = \bigcup_{r\,:\,Q_1(r)=i} C_a^r,$$

où l'union est prise sur l'ensemble des chaînes aléatoires r telles que $Q_1(r) = i$. Ensuite, nous associons à chaque position j de P_2 et réponse $b \in \Sigma_2$, un ensemble $S_{j,b}$. Si b ne satisfait pas l'ensemble des k clauses de ψ spécifiées aux positions $Q_2(r)$, alors $S_{j,b} = \varnothing$. Sinon,

$$S_{j,b} = \bigcup_{r\,:\,Q_2(r)=j} \overline{C}^r_{\pi(r,b)},$$

où l'union est prise sur l'ensemble des chaînes aléatoires r telles que $Q_2(r) = j$.

Soit r une chaîne aléatoire ; posons $i = Q_1(r)$ et $j = Q_2(r)$. Alors, les seuls ensembles de $\mathcal{S}$ qui contiennent des éléments de U^r sont :

- $S_{i,a}$, pour $a \in \Sigma_1$, et
- $S_{j,b}$, pour $b \in \Sigma_2$ tel que b satisfasse les k clauses spécifiées à la position j de P_2.

De plus, chaque ensemble du premier type contient exactement un ensemble parmi $C_1^r, \ldots, C_{2^k}^r$ et chaque ensemble du second type contient exactement un ensemble parmi $\overline{C}_1^r, \ldots, \overline{C}_{2^k}^r$.

Soit r une chaîne aléatoire et posons $i = Q_1(r)$ et $j = Q_2(r)$. Remarquons que $S_{i,a} \cup S_{j,b}$ couvre U^r ssi $\pi(r,b) = a$ et b satisfait les k clauses spécifiées à la position j de P_2. Soit $\mathcal{C}$ une couverture de $\mathcal{U}$. Si $\mathcal{C}$ contient une telle paire d'ensembles, nous dirons que $\mathcal{C}$ *contient une bonne couverture* pour U^r. Si $\mathcal{C}$ ne contient pas de bonne couverture pour U^r, alors, pour couvrir U^r, il doit contenir strictement plus de l ensembles de la forme $S_{i,a}$ ou $S_{j,b}$, avec $a \in \Sigma_1$ et $b \in \Sigma_2$. Dans ce cas, nous dirons que $\mathcal{C}$ *contient une mauvaise couverture* pour U^r.

Supposons que ϕ est satisfaisable. Alors, il existe une paire de preuves (y_1, y_2) telle que le vérificateur accepte avec probabilité 1. Construisons la couverture $\mathcal{C}$ suivante. Sélectionnons, pour chaque position i de P_1 et chaque position j de P_2, les ensembles $S_{i,a}$ et $S_{j,b}$, où a et b sont les réponses à ces questions dans y_1 et y_2 respectivement. Alors $|\mathcal{C}| = 2n^k$. On voit facilement que $\mathcal{C}$ contient une bonne couverture pour chaque ensemble U^r.

Supposons à présent que ϕ ne soit pas satisfaisable. V' rejette toute paire de preuves avec forte probabilité. Rappelons que nous avons supposé que les paramètres du gadget ont les valeurs désirées (ce qui arrive avec probabilité $> 1/2$). Soit $\mathcal{C}$ une couverture optimale de $\mathcal{U}$. Est-ce que $\mathcal{C}$ contient forcément une mauvaise couverture pour U^r sur la plupart des chaînes r ? Clairement, $\mathcal{C}$ peut sélectionner des ensembles correspondant à différents morceaux de plusieurs preuves. En utilisant cette propriété, ne peut-on pas construire une couverture qui ne soit qu'un tout petit peu plus grande que $2n^k$? Remarquons qu'un ensemble de $\mathcal{S}$ couvre des éléments de plusieurs univers U^r différents, ce qui complique la suite de l'argument.

Nous décrivons maintenant une procédure pour construire à partir de $\mathcal{C}$ une paire de preuves (y_1, y_2), telle que si $|\mathcal{C}|$ est petit, alors V' accepte cette paire avec forte probabilité. Nous pourrons alors en déduire la minoration désirée sur $|\mathcal{C}|$.

Considérons l'*ensemble des réponses* sélectionnées par $\mathcal{C}$ pour chaque position des deux preuves. Pour chaque position i de P_1 et pour chaque position j de P_2, posons $A(i) = \{a : S_{i,a} \in \mathcal{C}\}$ et $A(j) = \{b : S_{j,b} \in \mathcal{C}\}$. Nous construisons les preuves y_1 et y_2 en sélectionnant pour chaque position i de P_1 et j de P_2 un élément aléatoire de $A(i)$ et de $A(j)$ respectivement ; nous sélectionnons une réponse arbitraire si l'ensemble des réponses est vide à cette position. Posons $B_1 = \{r : |A(Q_1(r))| > l/2\}$, $B_2 = \{r : |A(Q_2(r))| > l/2\}$ et $G = \overline{B_1 \cup B_2}$.

Par construction, G est l'ensemble des chaînes aléatoires r telles que $\mathcal{C}$ sélectionne moins de $l/2$ réponses pour chacune des positions $Q_1(r)$ et $Q_2(r)$. La couverture optimale $\mathcal{C}$ contient donc une bonne couverture $S_{i,a} \cup S_{j,b}$ pour U^r, avec $a \in A(Q_1(r))$ et $b \in A(Q_2(r))$. La paire de preuves (y_1, y_2) contient a et b aux positions $Q_1(r)$ et $Q_2(r)$ respectivement, avec probabilité $\geqslant (\frac{2}{l})^2$. Ainsi, V' accepte les preuves (y_1, y_2) sur la chaîne aléatoire r avec au moins cette probabilité.

Soit f_G la fraction des chaînes aléatoires contenues dans G. D'après le théorème 29.26,

$$f_G \left(\frac{2}{l}\right)^2 \leqslant \mathbf{Pr}[V' \text{ accepte } \phi \text{ avec les preuves } (y_1, y_2)] \leqslant \delta^{dk}.$$

Ainsi, $f_G \leqslant \delta^{dk} l^2/4$. Comme l^2 vaut $O(\log^4 n)$, en prenant $k = O(\log \log n)$, nous assurons que $f_G < 1/2$. On en conclut que $B_1 \cup B_2$ contient au moins la moitié des chaînes aléatoires, et donc l'un des deux ensembles en contient au moins un quart. Notons B_i cet ensemble.

Par la propriété d'uniformité (remarque 29.22), si r est tiré uniformément au hasard, alors $Q_1(r)$ est une position aléatoire de P_1 et $Q_2(r)$ une position aléatoire de P_2 (même si elles sont corrélées) ; et r appartient à B_i avec probabilité $> 1/4$. Ainsi, les ensembles de réponses de plus d'un quart des positions de B_i sont de taille $> l/2$. Par conséquent, la taille de la couverture est $> ln^k/8 = \Omega(n^k k \log n)$. □

Une conséquence directe du théorème 29.27 est que le problème de la couverture par ensembles est inapproximable à moins que **NP** n'appartienne à la classe de complexité à erreur unilatérale[9] en temps $n^{O(\log\log n)}$. Des techniques classiques en théorie de la complexité (voir exercice 1.18) conduisent au résultat plus fort suivant.

Corollaire 29.29 *Il existe une constante b telle que s'il existe une $b\log n$-approximation pour le problème de couverture par ensembles de taille minimum, où n est la taille de l'ensemble univers de l'instance, alors* $\mathbf{NP} \subseteq \mathbf{ZTIME}(n^{O(\log\log n)})$ *(voir section A.4 pour la définition).*

29.8 Exercices

29.1 Montrez que $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$.
Indication : Soit $L \in \mathbf{PCP}(\log n, 1)$. La machine **NP** acceptant L devine la preuve, simule le vérificateur pour L sur toutes les chaînes de $O(\log n)$ bits, et accepte ssi le vérificateur accepte sur toutes les chaînes aléatoires.

29.2 Montrez que (définitions en annexe A) :

1. $\mathbf{PCP}(0,0) = \mathbf{PCP}(0, \log n) = \mathbf{P}$.
2. $\mathbf{PCP}(\text{poly}(n), 0) = \text{co-}\mathbf{RP}$, où $\text{poly}(n) = \bigcup_{k\geqslant 0} n^k$.
3. $\mathbf{PCP}(\log n, 1) = \mathbf{PCP}(\log n, \text{poly}(n))$.
   Indication : $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1) \subseteq \mathbf{PCP}(\log n, \text{poly}(n)) \subseteq \mathbf{NP}$.

29.3 Démontrez la réciproque du théorème 29.7, c'est-à-dire que s'il existe une réduction écartante de SAT à MAX-3SAT, alors $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$.
Indication : Réduisez toute formule SAT ϕ en une instance ψ de MAX-3SAT. Donnez une instanciation optimale de ψ pour preuve au vérificateur. La probabilité d'erreur est alors $1 - \varepsilon_M$. Répétez pour rendre la probabilité d'erreur $< 1/2$.

29.4 Exhibez une réduction écartante de MAX-3SAT(29) à MAX-3SAT(5) avec des paramètres convenables pour obtenir la difficulté de l'approximation du second problème.
Indication : La réduction est similaire à celle qui est mise en œuvre dans le théorème 29.11, quoique plus simple. Utilisez un cycle au lieu d'un expandeur. Toute instanciation inconsistante peut satisfaire jusqu'à 14 clauses, correspondant aux anciennes variables x. Mais elle doit laisser au moins deux clauses insatisfaites, correspondant aux arêtes du cycle de x.

[9] *One-sided-error complexity class*, en anglais.

29.5 Complétez la preuve du théorème 29.18.

29.6 (Håstad [130]) Une préoccupation importante pour caractériser **NP** par PCP, est la réduction du nombre de bits lus dans la preuve par le vérificateur. Le résultat remarquable suivant démontre que 3 bits suffisent.

Théorème 29.30 *Pour tout $\varepsilon > 0$,*

$$\mathbf{NP} = \mathbf{PCP}_{1-\varepsilon,\frac{1}{2}+\varepsilon}[\log n, 1].$$

De plus, il existe un vérificateur PCP particulièrement simple pour SAT. Il utilise $O(\log n)$ bits aléatoires pour calculer trois positions de la preuve i, j et k et un bit b, et accepte ssi

$$y(i) + y(j) + y(k) \equiv b \pmod 2,$$

où $y(i)$ désigne le i-ième bit de la preuve y.

1. Considérons la restriction du problème de la résolution d'équations linéaires sur GF[2] (problème 16.12, voir exercice 16.7), aux instances où chaque équation a exactement 3 variables. Utilisez la caractérisation du théorème 29.30 pour donner une réduction écartante appropriée de SAT à ce problème, qui montre que pour tout $\varepsilon > 0$, s'il existe une $(2-\varepsilon)$-approximation pour le second problème, alors $\mathbf{P} = \mathbf{NP}$.
2. Exhibez une réduction d'écart du problème de la résolution d'équations linéaires sur GF[2] à MAX-3SAT, qui montre que s'il existe une $(8/7-\varepsilon)$-approximation pour MAX-3SAT, avec $\varepsilon > 0$, alors $\mathbf{P} = \mathbf{NP}$.
   Indication : Transformez l'équation $x_i + x_j + x_k \equiv 0 \pmod 2$ en clauses comme suit :

$$(\overline{x}_i \vee x_j \vee x_k) \wedge (x_i \vee \overline{x}_j \vee x_k) \wedge (x_i \vee x_j \vee \overline{x}_k) \wedge (\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k).$$

29.7 Terminez la preuve du théorème 29.21, c'est-à-dire démontrez que $\mathbf{2P1R}_{1,1-\varepsilon_P}(\log(n)) \subseteq \mathbf{NP}$.

29.8 Démontrez les conditions uniformes introduites à la remarque 29.22.
Indication : Introduisez classiquement de nouvelles variables booléennes.

29.9 Prouvez le théorème 29.24 par la méthode probabiliste.
Indication : $p(m, 2^l) = O(m2^{2l})$ suffit. Sélectionnez chaque ensemble C_i en y incluant chaque élément de U aléatoirement et indépendamment avec probabilité 1/2.

29.10 (Feige [87]) Le résultat suivant donne des conditions plus fortes sur la difficulté de l'approximation de la couverture par ensembles :

Théorème 29.31 *Pour toute constante $\delta > 0$, s'il existe une $((1-\delta)\ln n)$-approximation pour le problème de la couverture par ensembles de taille minimum, où n désigne la taille de l'ensemble univers de l'instance, alors* $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$*, où* $\mathbf{DTIME}(t)$ *est la classe des problèmes qui se résolvent par un algorithme déterministe en temps $O(t)$.*

Considérons le problème de la couverture maximum (problème 2.18 de l'exercice 2.15). Démontrez à l'aide du théorème 29.31 que s'il existe pour un $\varepsilon > 0$ une $(1 - 1/e + \varepsilon)$-approximation pour le problème de la couverture maximum, alors $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$.
Indication : Utilisez une approximation pour le problème de la couverture maximum pour construire une $((1-\delta)\ln n)$-approximation pour la couverture par ensembles, pour un certain $\delta > 0$, comme suit. Devinez k, le nombre optimal d'ensembles nécessaires pour l'instance, et lancez l'algorithme de couverture maximum avec pour paramètre k, itérativement, jusqu'à ce qu'une couverture soit obtenue. À chaque itération, une fraction $(1 - 1/e + \varepsilon)$ des éléments non couverts sont couverts. Ainsi, le nombre d'itérations I vérifie : $(1/e - \varepsilon)^l = 1/n$.

29.11 (Jain, Mahdian, et Saberi [146]) En utilisant le théorème 29.31, montrez que s'il existe une $(1 + 2/e - \varepsilon)$-approximation pour le problème de la k-médiane métrique (problème 25.1) pour un $\varepsilon > 0$, alors $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)})$.

29.9 Notes

Le premier résultat sur la difficulté de l'approximation reposant sur les preuves vérifiables stochastiquement est dû à Feige, Goldwasser, Lovász, Safra et Szegedy [89]. Ces travaux lancèrent la quête du théorème PCP, fondée sur les travaux sur les systèmes de preuve interactive[10] (Babai [20] et Goldwasser, Micali et Rackoff [116]) et sur la vérification de programmes (Blum et Kannan [30], et Blum, Luby et Rubinfeld [31]). Le théorème PCP fut démontré par Arora et Safra [16], et Arora, Lund, Motwani, Sudan et Szegedy [16, 14]. Le théorème 29.30, qui conduit à des résultats de difficulté d'approximation optimaux pour plusieurs problèmes, est dû à Håstad [130].

Avant ces résultats, les travaux pionniers de Papadimitriou et Yannakakis [227] introduisirent la notion de **Max-SNP**-complétude qui mit en évidence la difficulté de l'approximation de plusieurs problèmes naturels. Leurs L-réductions sont plus contraignantes que les réductions d'écart. Par conséquent, les idées motrices de leurs réductions se transportent directement dans le nouveau cadre : les réductions des théorèmes 29.11 et 29.13 en sont des exemples. En fait, une des motivations originelles du théorème PCP était qu'obtenir un résultat sur la difficulté de l'approximation de MAX-SAT

[10] *Interactive proof systems*, en anglais.

entraînerait directement des résultats de difficulté pour l'approximation de tous les problèmes **Max-SNP**-difficiles. Le théorème 29.14 est de Bern et Plassmann [27].

Lubotzky, Phillips et Sarnak [205] sont à l'origine de la construction des expandeurs. Le théorème 29.17 est dû à Impagliazzo et Zuckerman [143]. Le théorème 29.19 sur la difficulté de l'approximation de Clique est tiré de [89] et [16, 14]. Le meilleur résultat actuel sur Clique, dû à Håstad [129], affirme qu'elle ne peut être approchée à un facteur $n^{1-\varepsilon}$ pour aucun $\varepsilon > 0$, à moins que **NP** = **ZPP**. Cette borne est très proche du meilleur facteur d'approximation $O(n/(\log^2 n))$ obtenu par Boppana et Holldórsson [32].

Lund et Yannakakis [207] donnèrent le premier résultat sur la difficulté de l'approximation de la couverture par ensembles, en démontrant que, à moins que $\mathbf{NP} \subseteq \mathbf{ZTIME}(n^{O(\text{polylog } n)})$, elle ne peut être approchée à un facteur $\log n/2$. L'amélioration de ce résultat, présenté au théorème 29.31, est due à Feige [87]. Cette amélioration repose sur l'utilisation de systèmes de preuve à k prouveurs. Naor, Schulman et Srinivasan [220] proposèrent une construction déterministe du système d'ensembles gadget du théorème 29.24, permettant ainsi de remplacer **ZTIME** par **DTIME** dans la conséquence sur les classes de complexité. Le système de preuve à deux prouveurs en une ronde fut défini par Ben-Or, Goldwasser, Kilian et Wigderson [26]. Le théorème 29.26 est dû à Raz [240].

Karloff et Zwick [170] obtinrent une 8/7-approximation pour la restriction de MAX-3SAT aux instances satisfaisables. Ceci complète le résultat sur la difficulté de l'approximation de ce problème présenté dans l'exercice 29.6.

Pour plus d'information sur ce thème, référez-vous à l'état de l'art d'Arora et Lund [13]. L'excellent compendium accessible sur le site

`http://www.nada.kth.se/~viggo/problemlist/compendium.html`

maintient à jour le statut des meilleurs résultats connus, positifs et négatifs, pour de très nombreux problèmes d'optimisation **NP**-difficiles. Ce compendium a également été publié par Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela et Protasi [19].

30 Problèmes ouverts

Ce chapitre fait le tour des problèmes et questions actuellement en vogue dans le domaine des algorithmes d'approximation De nouvelles problématiques importantes apparaîtront dans le futur. Deux questions se posent naturellement pour chacun des problèmes présentés ci-dessous : réaliser le meilleur facteur d'approximation et obtenir un résultat de difficulté d'approximation correspondant.[1].

30.1 Problèmes ayant un algorithme à un facteur constant

Comme un grand nombre de questions ouvertes du domaine consistent aujourd'hui à améliorer le facteur d'approximation pour des problèmes qui admettent déjà des approximations à un facteur constant, nous avons préféré leurs consacrer une section à part. Bien sûr, le but n'est pas d'optimiser localement les techniques utilisées pour améliorer petit à petit le facteur d'approximation. Un bon exemple est l'amélioration réalisée par Goemans et Williamson du facteur d'approximation pour MAX-CUT, de 1/2 à 0.878, qui ajoutèrent la programmation semi-définie aux outils du domaine. L'étude de la plupart des problèmes ci-dessous a le potentiel d'enrichir significativement des techniques connues *via* des idées nouvelles et importantes.

Couverture par sommets, problème 1.1 : Améliorer le facteur d'approximation 2 (voir algorithmes des chapitres 1, 2, 14, et 15). La programmation semi-définie est une direction possible, voir la tentative de Goemans et Kleinberg [110].

Couverture par ensembles, problème 2.1 : Cette question est une généralisation de la précédente. Considérez la restriction du problème de la couverture par ensembles aux instances dans lesquelles la fréquence de chaque élément est inférieure à une constante f donnée. Améliorer le

[1] Consulter l'excellente traité
`http://www.nada.kth.se/~viggo/problemlist/compendium.html`
pour un état de l'art mis à jour régulièrement des meilleurs résultats positifs et négatifs pour de nombreux problèmes d'optimisation **NP**-difficiles.

facteur f (voir algorithmes des chapitres 2, 14, et 15). Le meilleur résultat de difficulté de l'approximation connu pour ce problème est $f^{1/19}$, en supposant que $\mathbf{P} \neq \mathbf{NP}$, par Trevisan [256].

Sous-graphe sans cycle maximum, problème 1.9 : Améliorer le facteur 1/2 (voir exercice 1.1). La programmation semi-définie pourrait s'appliquer.

TSP métrique, problème 3.5 : Comme cela est relaté dans l'exercice 23.13, la solution générée par l'algorithme de Christofides (algorithme 3.10) est à un facteur 3/2 de la relaxation linéaire par élimination des sous-tours pour ce problème. Cependant, le pire exemple de saut intégral connu est 4/3. Existe-t-il une 4/3-approximation utilisant cette relaxation ?

L'algorithme de Christofides fonctionne en deux étapes : construire un MST puis corriger la parité des degrés des sommets. Le résultat énoncé ci-dessus s'obtient en bornant ces deux étapes indépendamment. Une bonne idée serait de commencer par étudier une 3/2-approximation en comparant directement la solution entière à la relaxation linéaire. Le schéma primal-dual pourrait en être la clé.

L'arbre de Steiner, problème 3.1 : Le meilleur facteur d'approximation connu est 5/3 (voir exercice 22.12). Une voie prometteuse pour améliorer ce facteur est l'emploi de la relaxation par coupes orientées (22.7). Cette relaxation est exacte pour l'arbre couvrant de poids minimum. Pour l'arbre de Steiner, le pire saut intégral connu est 8/7, dû à Goemans (voir exercice 22.11). La meilleure majoration connue du saut intégral est 3/2 pour les graphes quasiment bipartis (c'est-à-dire les graphes qui ne contiennent pas d'arêtes reliant deux sommets Steiner), due à Rajagopalan et Vazirani [235]. Le problème est de déterminer le saut intégral de cette relaxation et d'obtenir un algorithme qui le réalise[2].

Rappelons que le saut intégral de la relaxation linéaire (22.2) est connu et vaut 2, non seulement pour ce problème, mais également pour le cas particulier de l'arbre couvrant de poids minimum et sa généralisation, le problème du réseau de Steiner.

Réseau de Steiner, problème 23.1 : Le chapitre 23 présente une 2-approximation. Elle utilise cependant l'arrondi en programmation linéaire et a un temps d'exécution prohibitif. La question est d'obtenir une 2-approximation combinatoire pour ce problème. Une conséquence de l'analyse de l'algorithme 23.7 est que le saut intégral de la relaxation linéaire (23.2) est inférieur à 2. Cette relaxation peut donc être utilisée comme un minorant pour obtenir une 2-approximation combinatoire. Le schéma primal-dual semble être une voie intéressante. Un point de départ serait

[2] La question plus générale sous-jacente est de clarifier les liens mystérieux entre le saut intégral d'une relaxation linéaire et le facteur d'approximation qu'on peut en tirer.

de déterminer si le fait suivant est vrai : « quelle que soit l'instance de la forêt de Steiner (et plus général du réseau de Steiner), il existe une solution primale entière $\boldsymbol{x}$ et une solution duale réalisable $\boldsymbol{y}$ telles que chaque arête sélectionnée dans $\boldsymbol{x}$ est saturée dans la solution duale $\boldsymbol{y}$ et que le degré de chaque dual levé S est $\leqslant 2$ ($\leqslant 2f(S)$) ». Observez que le degré de la solution duale calculée par l'algorithme 22.3 peut être arbitrairement grand.

Coupe multiséparatrice,[3] **problème 4.1 :** Le chapitre 19 présente une 1.5-approximation. Comme nous l'avons dit, ce facteur peut être amélioré à 1.3438. Cependant, le pire exemple de saut intégral connu pour la relaxation linéaire (19.1) est 8/7. La question est de déterminer le saut intégral de cette relaxation, et d'obtenir un algorithme offrant cette garantie. Une autre relaxation est proposée dans l'exercice 19.7. Quels liens existent-ils entre ces deux relaxations ? Sont-elles équivalentes au sens qu'une solution réalisable de l'une peut être transformée en une solution réalisable de l'autre de même coût ?

Coupe-cycles-distingués de sommets,[4] **problème 19.15 :** Le meilleur facteur d'approximation connu est 8, par un algorithme assez compliqué (voir exercice 19.13). Existe-t-il une 2-approximation, réalisant ainsi la même garantie que pour d'autres problèmes similaires évoqués à l'exercice 19.13 ?

30.2 Autres problèmes d'optimisation

Vecteur le plus court, problème 27.1 : Il s'agit d'obtenir une approximation à un facteur polynomial pour ce problème. Comme nous l'avons vu au chapitre 27, on peut obtenir avec le module dual un certificat co-**NP** à un facteur n pour ce problème. Le module dual a-t-il une utilité plus avancée en algorithmique ? Est-ce qu'une permutation aléatoire des vecteurs de la base aiderait avant de lancer l'algorithme 27.9 ? Le meilleur résultat de difficulté de l'approximation pour ce problème, d'un facteur $\sqrt{2}-\varepsilon$ pour tout $\varepsilon > 0$ en supposant que $\mathbf{RP} \neq \mathbf{NP}$, est dû à Micciancio [213].

Coupe la moins dense, problème 21.2 : Le meilleur facteur d'approximation connu est $O(\log n)$ (voir chapitre 21). Cependant, aucun résultat sur la difficulté de l'approximation n'a été obtenu pour ce problème — à notre connaissance, un PTAS n'est pas encore écarté. Existe-t-il une approximation à un facteur constant, voire un PTAS pour ce problème ?

[3] *Multiway cut*, en anglais.
[4] *Subset feedback vertex set*, en anglais.

Coupe b-équilibrée minimum et bisection minimum, problème 21.27 : Feige et Krauthgamer [90] ont obtenu une $O(\log^2 n)$-approximation pour ces deux problèmes. Comme pour la coupe la moins dense, l'existence d'un PTAS n'a toujours pas été écartée. Existe-t-il une approximation à un facteur constant, voir un PTAS pour ces problèmes ? Limité aux graphes planaires, le problème de la coupe b-équilibrée minimum admet une 2-approximation lorsque $b \leqslant 1/3$, voir Garg, Saran et Vazirani [102].

Multicoupe minimum, problème 18.1 : Nous avons donné une $O(\log n)$-approximation au chapitre 20. Une question ouverte depuis de longues années est de statuer sur l'existence d'une approximation à un facteur constant déterministe pour ce problème.

TSP asymétrique, problème 3.15 : Le meilleur facteur d'approximation connu est $O(\log n)$ (voir exercice 3.6). Existe-t-il une approximation à un facteur constant pour ce problème ?

Conception de réseau sommet-connexe : Dans cette variante du problème du réseau de Steiner (problème 23.1), il s'agit de trouver un sous-graphe de coût minimum contenant $r_{u,v}$ chemins sommet-disjoints, au lieu d'arêtes-disjoints, entre chaque paire de sommets $u, v \in V$. Aucun algorithme non trivial n'est connu pour cette variante. Dans le cas particulier où $r_{u,v} = k$ pour tout $u, v \in V$ et où les coûts des arêtes sont métriques, Khuller et Raghavachari [178] proposèrent une $(2 + \frac{2(k-1)}{n})$-approximation. Un problème de difficulté intermédiaire est la conception d'un réseau élément-connexe, où les sommets sont partitionnés en deux ensembles : les terminaux et les non terminaux. Seuls les arêtes et les non terminaux, que nous appellerons les *éléments*, peuvent tomber en panne. Les contraintes de connectivité ne portent que sur les paires de terminaux, et demandent un certain nombre de chemins élément-disjoints pour chaque paire. Jain, Măndoiu, Vazirani et Williamson [147] donnèrent une $2H_k$-approximation, où k est la valeur de la plus grande demande.

Multiflot entier maximum, problème 18.3 : L'exemple 18.8 montre que le saut intégral la relaxation linéaire naturelle vaut $\Omega(n)$. On contourne facilement cette difficulté tout en conservant l'essence du problème original, en recherchant un flot maximum demi-entier. Existe-t-il une $O(\log n)$-approximation pour ce dernier problème ?

Placement d'installations métrique sans capacité[5] et k-médiane, problèmes 24.1 et 25.1 : Déterminer les sauts intégrales des relaxations linéaires (24.2) et (25.2).

Placement d'installations métrique avec capacités,[6] exercice 24.8 : Comme l'affirme l'exercice 24.8, le saut intégral de l'adaptation du pro-

[5] *Uncapacitated facility location problem*, en anglais.
[6] *Capacitated facility location problem*, en anglais.

gramme linéaire (24.2) à ce problème est arbitrairement grand. Existe-t-il une autre façon de minorer qui donne un bon algorithme d'approximation ?

Multicoupe et coupe la moins dense orientées : Dans les chapitres 20 et 21, nous avons étudié deux généralisations du problème du flot maximum non orienté et proposé des algorithmes d'approximation pour les problèmes de coupes correspondants, la multicoupe et la coupe la moins dense. On ne connaît pas grand chose à ce jour sur les variantes de ces problèmes sur les graphes orientés.

Arborescence de Steiner, problème 3.14 : Comme l'affirme l'exercice 3.3, ce problème a peu de chance d'admettre un algorithme d'approximation à un facteur $O(\log n)$. La garantie $O(\log n)$ est-elle possible ? Le meilleur facteur d'approximation connu est n^{ε} pour tout $\varepsilon > 0$ donné, par Charikar et. al. [39]. Il faudrait également étudier les généralisations de ce problème avec des contraintes de connectivité plus fortes, similaires au problème du réseau de Steiner.

Coupe-circuits d'arêtes (et de sommets)[7] : Étant donné un graphe orienté $G = (V, E)$, un *coupe-circuits d'arêtes (de sommets)* est un ensemble d'arêtes (de sommets) dont le retrait rend le graphe acyclique. Le problème est de trouver un tel ensemble de taille minimum. La variante pondérée assigne un poids à chaque arête (sommet), et il s'agit de trouver un coupe-circuits de poids minimum. On peut voir facilement que ces deux problèmes se réduisent l'un à l'autre via des réductions isofacteur. Seymour [247] proposa une $O(\log n \log\log n)$-approximation pour la version pondérée. Peut-on porter le facteur d'approximation à $O(\log n)$ voire même à une constante ?

Temps de couverture[8] : Étant donné un graphe non orienté $G = (V, E)$, le temps de couverture $C(v)$ depuis un sommet $v \in V$ est l'espérance du nombre de pas effectués par une marche aléatoire sur G qui commence en v et visite tous les sommets. Le *temps de couverture* de G est défini comme $\max_{v \in V} C(v)$. Clairement, un algorithme randomisé peut estimer le temps de couverture avec une précision arbitraire, en prenant la moyenne de nombreuses simulations empiriques de la marche aléatoire. Kahn, Kim, Lovász et Vu [159] proposèrent une $O((\log\log n)^2)$-approximation déterministe pour ce problème. Existe-t-il une approximation déterministe à un facteur constant ?

[7] *Directed feedback edge (vertex) set*, en anglais.
[8] *Cover time*, en anglais.

30.3 Problèmes de dénombrement

Les problèmes de décision associés aux problèmes ci-dessous sont dans **P**, et (à l'exception des triangulations et des graphes ayant une séquence de degrés fixée) le dénombrement de leurs solutions est #**P**-complet, et la complexité du dénombrement approximatif de leurs solutions est ouverte. Les complexités du comptage des graphes avec une séquence de degrés fixée ou des triangulations sont ouvertes, bien que conjecturées #**P**-complètes.

Couplages parfaits dans les graphes généraux : Restreint aux graphes planaires, ce problème se résout en temps polynomial avec l'algorithme classique de Kastelyn [176]. Ce résultat s'étend aux graphes sans $K_{3,3}$ (c'est-à-dire les graphes qui n'ont pas de sous-graphe homéomorphe à $K_{3,3}$), voir Little [199] et Vazirani [260]. Il existe un FPRAS pour la restriction de ce problème aux graphes bipartis, problème équivalent à l'évaluation d'un permanent à coefficients dans $\{0,1\}$, dû à Jerrum, Sinclair et Vigoda [151] (plus généralement, leur résultat donne un FPRAS pour évaluer le permanent de toute matrice à coefficients entiers positifs).

Volume d'un objet convexe : Étant donné un objet convexe de $\mathbf{R}^n$ via un oracle, le problème est d'estimer son volume. Plusieurs problèmes de dénombrement se ramènent à ce problème fondamental. Un premier FPRAS pour ce problème fut proposé par Dyer, Frieze et Kannan [74]. Bien que polynomial, son temps d'exécution était exorbitant. Il faisait $O^*(n^{23})$ appels à l'oracle – la notation O^* signifie que les facteurs en $\log n$ et la dépendance en ε (la borne sur l'erreur) ne sont pas pris en compte. Le meilleur algorithme actuel, dû à Kannan, Lovász et Simonovits [163], requiert $O^*(n^5)$ appels à l'oracle, et $O^*(n^7)$ opérations arithmétiques. Peut-on continuer d'améliorer le temps d'exécution ?

Orientations acycliques : Dénombrer les orientations acycliques d'un graphe non orienté G donné. Une orientation des arêtes d'un graphe G est dite *acyclique* si le graphe orienté résultant est acyclique. On connaît plusieurs chaînes de Markov sur l'ensemble des orientations acycliques qui convergent vers une distribution uniforme ; cependant, on ne sait pas si l'une d'elles est rapidement mélangeante. Par exemple, considérons que deux orientations sont voisines si l'on obtient l'une à partir de l'autre en inversant les orientations des arêtes incidentes à une source ou à un puits, où une source (un puits) est définie par un degré entrant (sortant) nul, et effectuons une marche aléatoire sur le graphe de voisinage correspondant.

Forêts : Une forêt d'un graphe non orienté est un ensemble d'arêtes sans cycle. Une forêt maximale est un arbre couvrant (en supposant que le graphe est connexe). Il est intéressant de noter que le dénombrement des arbres couvrants d'un graphe est dans **P** – c'est l'un des rares problèmes de dénombrement qu'on sait résoudre en temps polynomial.

Ce résultat est une conséquence du théorème classique de la matrice d'arbre de Kirchhoff, voir [202]. Remarquons également qu'il existe des algorithmes polynomiaux très élégants pour générer un arbre couvrant aléatoire d'un graphe non orienté à l'aide de chaînes de Markov rapidement mélangeantes, dus à Aldous [5], Broder [36] et Wilson [268]. Cependant, la complexité du dénombrement des forêts de graphes arbitraires est ouverte. Le cas des graphes denses (où le degré de chaque sommet est supérieur à αn, avec $0 < \alpha < 1$) est traité par Annan [10]. Les forêts et les arbres couvrants sont respectivement les familles de vecteurs linéairement indépendants et les bases du matroïde graphique d'un graphe donné.

Bases d'un matroïde : Dénombrer les bases d'un matroïde arbitraire, donné par un oracle d'indépendance. Nous définissons le *graphe d'échange des bases* d'un matroïde comme suit : les bases en sont les sommets, et deux bases sont adjacentes ssi leur différence symétrique est une paire de deux éléments. Dagum, Luby, Mihail et Vazirani [60] ont conjecturé que la chaîne de Markov définie par la marche aléatoire sur le graphe d'échange des bases mélange rapidement. Si tel était le cas, on pourrait en déduire un FPRAS pour dénombrer approximativement les bases. Des exemples de matroïdes pour lesquels cette conjecture a été prouvée sont les matroïdes graphiques (voir le problème précédent) et leur généralisation, les matroïdes équilibrés. Reportez-vous à Feder et Mihail [85] pour ce résultat. Une réponse positive à cette question permettrait également le dénombrement approximatif des forêts (puisque les forêts d'une taille donnée sont les bases du matroïde graphique tronqué convenablement).

Fiabilité d'un réseau : Plusieurs variantes de ce problème se sont révélées utiles en pratique et ont été étudiées par le passé, en particulier, deux variantes sur les graphes non orientés : la fiabilité entre s et t, qui demande la probabilité qu'une paire de sommets s et t donnée soit déconnectée, et la fiabilité globale, qui demande la probabilité qu'une partie du graphe soit déconnectée. On peut définir des problèmes similaires sur les graphes orientés. De ces quatre problèmes, seule la fiabilité globale semble cernée — nous en avons présenté un FPRAS au chapitre 28. On pourrait également chercher à évaluer dans les quatre cas la probabilité que la paire s–t ou que le graphe entier restent connectés. Cette variante est ouverte même pour les graphes non orientés.

Tours eulériens : Dénombrer les tours eulériens d'un graphe non orienté donné (un graphe connexe est eulérien ssi tous ses sommets sont de degré pair). Il est intéressant de noter qu'il existe un algorithme polynomial pour compter les circuits eulériens d'un graphe orienté – c'est encore une conséquence du théorème de Kirchhoff.

Arbres : Dénombrer dans un graphe non orienté G donné, les sous-graphes qui sont des arbres.

Antichaîne d'un ordre partiel : voir exercice 1.7 pour la définition. Il existe un FPRAS, dû à Matthews [211], Karzanov et Khachian [175] et Bubley et Dyer [37], pour le problème assez similaire du dénombrement des ordres totaux compatibles avec un ordre partiel donné.

Graphes ayant une séquence de degrés fixée : Dénombrer les graphes simples dont les degrés des sommets $v_1, \ldots, v_n$ forment une séquence $d_1, \ldots, d_n$ donnée. Un problème voisin est le dénombrement des graphes connexes ayant une séquence de degrés fixée. Pour ces deux problèmes, on peut déterminer en temps polynomial si un tel graphe existe, à l'aide d'un algorithme de couplage. Il existe un FPRAS lorsqu'on se restreint aux graphes bipartis, avec une bipartition fixée, en utilisant le résultat sur les permanents à coefficients dans $\{0, 1\}$ [151].

Tomographie 2D discrète[9] : Dénombrer les matrices $m \times n$ à coefficients entiers positifs dont les sommes des coefficients de chaque ligne et de chaque colonnes sont des constantes fixées. Dyer, Kannan et Mount [71] proposèrent un FPRAS pour les cas où les sommes sur chaque ligne et chaque colonne sont suffisamment grandes, supérieures à $(m+n)mn$. Morris [217] étendit ce résultat au cas où les sommes sur chaque ligne valent $\Omega(n^{3/2} m \log m)$ et les sommes sur chaque colonne valent $\Omega(m^{3/2} n \log n)$. Si les matrices sont à coefficients dans $\{0, 1\}$, ce problème est identique à celui du dénombrement des graphes bipartis ayant une séquence de degrés fixée, pour lequel nous avons un FPRAS basé sur les permanents à coefficients dans $\{0, 1\}$ [151].

Triangulations : Dénombrer les triangulations de n points du plan, c'est-à-dire le nombre de façons de relier les points par des segments ne s'intersectant pas, de sorte que toutes les faces internes soient des triangles. Considérez le graphe G dont les sommets sont toutes les triangulations possibles, et tel que deux triangulations t et t' sont adjacentes si t' s'obtient à partir de t en ôtant une arête d'une face finie de t telle que la face devienne un quadrilatère convexe, puis en y rajoutant l'arête reliant les deux autres sommets du quadrilatère. On conjecture que la marche aléatoire sur ce graphe mélange rapidement. Si les n points forment les sommets d'un polygone convexe, alors le nombre de triangulations est connu, c'est le nombre de Catalan C_{n-2}, et se calcule donc en temps polynomial. On sait que dans ce cas particulier, la chaîne de Markov mélange rapidement, voir McShine et Tetali [212].

Mariages stables : Une instance du problème du mariage stable est la donnée de n garçons, de n filles, et des listes ordonnées des préférences de chaque garçon et de chaque fille (chaque fille ordonne totalement les n garçons et vice versa). Un mariage est un couplage parfait des filles et des garçons. Un garçon g et une fille f forment un *couple tentateur* s'ils ne

[9] *Contingency tables*, en anglais.

sont pas mariés ensemble et si g préfère f à la fille à laquelle il est marié, et si f préfère g au garçon auquel elle est mariée. Un mariage est *stable* s'il ne contient aucun couple tentateur. On ne connaît pas la complexité du dénombrement approximatif des mariages stables. Gusfield et Irving présentent dans leur livre [127] les nombreuses propriétés structurelles de l'ensemble des mariages stables.

Colorations d'un graphe : Considérons un graphe non orienté $G = (V, E)$ de degré maximum Δ. Jerrum [152] donna un FPRAS pour dénombrer les k-colorations valides de G pour tout $k > 2\Delta$, et Vigoda [263] étendit ce résultat à tout $k > 11\Delta/6$. Peut-on étendre ce résultat au dénombrement de k-colorations de G pour tout $k \geqslant \Delta + 2$? Notons que, si le nombre de couleurs est $\leqslant \Delta + 1$, alors la chaîne de Markov naturelle qui à chaque étape tire un sommet uniformément et le recolore avec une couleur aléatoire cohérente, n'est plus nécessairement connexe. Cette quantité a des applications en physique statistique.

Cycles hamiltoniens : Si tous les sommets d'un graphe non orienté G ont un degré $\geqslant n/2$ alors G contient un cycle hamiltonien (voir la condition de Dirac dans [202]). Dyer, Frieze et Jerrum [72] donnèrent un FPRAS pour le cas où le degré minimum est $\geqslant (1/2+\varepsilon)n$, pour un $\varepsilon > 0$. Peut-on étendre ce résultat au cas $\varepsilon = 0$, c'est-à-dire aux graphes dont le degré minimum est $n/2$?

Stables (ensembles indépendants) : Luby et Vigoda [206] proposèrent un FPRAS pour les graphes de degré maximum $\Delta = 4$. Dyer, Frieze et Jerrum [73] démontrèrent que ce problème n'est pas approximable si $\Delta \geqslant 25$, à moins que $\mathbf{RP} = \mathbf{NP}$. Ils développèrent également un argument démontrant que la chaîne de Markov Monte Carlo a peu de chance de marcher dès que $\Delta \geqslant 6$. En dehors du cas $\Delta = 5$, il reste à déterminer si d'autres méthodes peuvent fonctionner pour $6 \leqslant \Delta \leqslant 24$, ou bien si ces cas sont également inapproximables.

Polynôme de Tutte : Plusieurs des problèmes évoqués ci-dessus sont des cas particuliers d'évaluation du polynôme de Tutte d'un graphe $G = (V, E)$ donné en des points particuliers (x, y) du plan. Pour tout $A \subseteq E$, on appelle *rang* de A, la quantité $r(A) = |V| - k(A)$, où $k(A)$ désigne le nombre de composantes connexes du sous-graphe ayant pour sommets V et arêtes A. Le *polynôme de Tutte* de G au point (x, y) est

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)}.$$

Ce polynôme engendre plusieurs quantités naturelles :
– en $(1, 1)$, T compte le nombre d'arbres couvrants de G.
– en $(2, 1)$, T compte le nombre de forêts de G.
– en $(1, 2)$, T compte le nombre de sous-graphes connexes of G.

– en $(2,0)$, T compte le nombre d'orientations acycliques de G.
– en $(0,2)$, T compte le nombre d'orientations de G qui engendrent un graphe orienté fortement connexe.
– le *polynôme chromatique* de G est donné par

$$P(G,\lambda) = (-1)^{r(E)}\lambda^{k(E)}T(G;1-\lambda,0),$$

où $P(G,\lambda)$ est le nombre de colorations de G avec λ couleurs.
– si la probabilité de panne de chaque arête est p, alors la probabilité que G reste connexe est donnée par :

$$R(G;p) = q^{|E|-r(E)}p^{r(E)}T(G;1,1/(1-p)).$$

Vertigan et Welsh [262] démontrèrent qu'en dehors de quelques points particuliers et de deux hyperboles particulières (voir le paragraphe suivant pour plus de précisions), l'évaluation exacte du polynôme de Tutte est #**P**-difficile. La question d'un FPRAS est grande ouverte. Annan [9] et Alon, Frieze et Welsh [7] donnèrent des FPRAS pour les graphes α-denses dans les cas $y = 1, x \geqslant 1$ et $y > 1, x \geqslant 1$, respectivement (un graphe est α-dense si chacun de ses sommets est de degré $\geqslant \alpha n$, avec $0 < \alpha < 1$)

Fonctions de partition des modèles d'Ising et de Potts :
L'hyperbole H_α définie par

$$H_\alpha = \{(x,y) : (x-1)(y-1) = \alpha\}$$

joue un rôle particulier dans le polynôme de Tutte. En particulier, le long de H_2, T donne la fonction de partition du modèle d'Ising sur G, et le long de H_Q, pour un entier $Q \geqslant 2$, T donne la fonction de partition du modèle de Potts sur G. Ces deux quantités sont utilisées en physique statistique ; reportez-vous à Welsh [265] pour les définitions exactes et plus de précisions (les points de chaque hyperbole représentent les différentes « températures » et Q est le nombre de classes de « couleurs »). Jerrum et Sinclair [154] proposèrent un FPRAS pour estimer la fonction de partition du modèle d'Ising d'un graphe à toute température, et Randall et Wilson [237] étendirent ce résultat en proposant une procédure d'échantillonnage en temps polynomial. Cependant, les exposants des temps de calcul de ces algorithmes sont bien trop élevés pour être utiles en pratique. Le processus de Swendsen-Wang [254] donne une façon naturelle d'estimer ces quantités avec des chaînes de Markov. La question est donc de déterminer si cette chaîne mélange rapidement. Un résultat négatif a été obtenu par Gore et Jerrum [119] qui démontrèrent que cette chaîne ne mélange pas rapidement sur les graphes complets K_n, pour $Q \geqslant 3$. Des résultats positifs furent obtenus pour différentes classes de graphes par Cooper et Frieze [56]. Cette chaîne mélange-t-elle rapidement pour la fonction de

partition du modèle d'Ising sur un graphe arbitraire ? Existe-t-il d'autres moyens d'estimer la fonction de partition du modèle de Potts sur un graphe arbitraire ?

30.4 Notes

Depuis la parution de cette liste de problèmes dans la première édition, de nouvelles avancées ont été accomplies. Dinur, Guruswami, Khot et Regev [65] ont démontré que pour tout $\varepsilon > 0$ constant, il n'existe pas de $(f - 1 - \varepsilon)$-approximation pour le problème de la couverture par ensembles (problème 2.1) pour les instances dont la fréquence de chaque élément est inférieure à f, pour tout $f \geqslant 3$ fixé, à moins que $\mathbf{P} = \mathbf{NP}$. Chekuri, Gupta et Kumar [44] ont montré que le saut intégral de la relaxation de l'exercice 19.7 est strictement pire que celui de la relaxation (19.1) pour le problème de la coupe multiséparatrice,[10] problème 4.1. Cryan, Dyer, Goldberg, Jerrum et Martin [59] ont donné un FPRAS pour le problème du dénombrement des solutions de la tomographie 2D discrète[11] lorsque le nombre de lignes est constant, à l'aide de chaînes de Markov Monte Carlo. Dyer [70] a obtenu également un algorithme plus efficace par programmation dynamique.

[10] *Multiway cut*, en anglais.
[11] *Contingency table with given row and column sums*, en anglais.

Annexes

A Éléments de théorie de la complexité

A.1 Certificats et classe NP

Un *problème de décision* est un problème dont la réponse est « oui » ou « non ». En voici deux exemples :

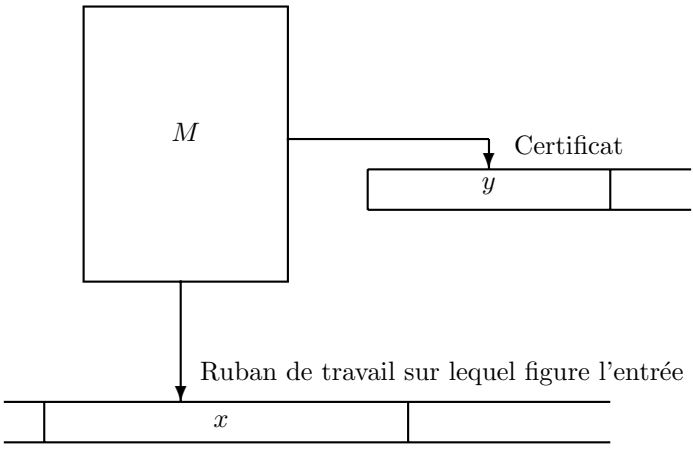
SAT : Étant donné une formule booléenne f sous forme normale conjonctive, existe-t-il une instanciation qui satisfasse f ?

Couverture par sommets de taille minimum : Étant donné un graphe non orienté G et un entier k, G admet-il une couverture par sommets de taille $\leqslant k$?

Nous appellerons kSAT la restriction de SAT aux instances dont toutes les clauses ont au plus k littéraux.

Il sera pratique de voir un problème de décision comme un langage, c'est-à-dire un sous-ensemble de $\{0,1\}^*$. Ce langage est l'ensemble des mots codant des instances positives du problème de décision. Nous dirons qu'un langage L appartient à **NP** s'il existe un polynôme p et une machine de Turing M de temps polynomial, appelée le *vérificateur*, telle que pour tout $x \in \{0,1\}^*$:

- si $x \in L$, alors il existe un mot y (le certificat) tel que $|y| \leqslant p(|x|)$, et que $M(x,y)$ accepte, et
- si $x \notin L$, alors pour tout mot y, avec $|y| \leqslant p(|x|)$, alors $M(x,y)$ rejette.



Le mot y qui aide à affirmer que x est une instance « positive », est un *certificat positif* de x. Nous appellerons également y *preuve* ou *solution* ; y sera aussi parfois appelé *témoin* dans le contexte des calculs randomisés. La classe **NP** est donc la classe des langages qui admettent des certificats positifs « courts et rapidement vérifiables ».

Prenons un exemple. Un vérificateur pour la couverture par sommets de taille minimum présume que y code un ensemble de sommets. Il vérifie si ce sous-ensemble est bien une couverture par sommets et que sa taille est bien inférieure à la borne — observez qu'aucune hypothèse n'a été faite sur le temps nécessaire pour *trouver* un tel certificat. Il est facile de voir que la classe **NP** ainsi définie est exactement la classe des langages reconnus par les machines de Turing non déterministes en temps polynomial (voir les références section A.6), d'où son nom.

Un langage L appartient à la classe co-**NP** ssi $\overline{L} \in \mathbf{NP}$. La classe co-**NP** est donc la classe des langages qui admettent des certificats négatifs « courts et rapidement vérifiables ». Considérons par exemple, le langage L des nombres premiers. Ce langage admet des certificats négatifs : une factorisation non triviale d'un nombre n est une preuve que $n \notin L$. Ainsi, $L \in$ co-**NP**. De façon intéressante, $L \in \mathbf{NP}$ également (voir exercice 1.13). En fait, il a été démontré en 2002 par Agrawal, Kayal et Saxena [2] que L appartient à **P**.

A.2 Réductions et NP-complétude

Nous introduisons maintenant la notion cruciale de *réduction polynomiale*. Soient L_1 et L_2 deux langages de **NP**. Nous dirons que L_1 *se réduit à* L_2, que nous noterons $L_1 \preceq L_2$, s'il existe une machine de Turing T polynomiale qui sur toute chaîne $x \in \{0,1\}^*$, renvoie une chaîne y telle que $x \in L_1$ ssi $y \in L_2$. En général, il n'est pas nécessaire de décider si x est une instance positive ou négative pour construire y. Clairement, si $L_1 \preceq L_2$ et si L_2 est décidable en temps polynomial, alors L_1 aussi.

Un langage L est **NP**-*difficile* si pour tout langage $L' \in \mathbf{NP}$, $L' \preceq L$. Un langage L est **NP**-*complet* si $L \in \mathbf{NP}$ et L est **NP**-difficile. Un langage **NP**-complet L est un langage le plus difficile de **NP**, au sens que s'il existe un algorithme polynomial pour L, alors il existe un algorithme polynomial pour tous les langages de **NP**, c'est-à-dire $\mathbf{P} = \mathbf{NP}$.

Le théorème central de la théorie de la complexité prouve qu'un problème naturel, SAT, est **NP**-complet. L'idée de la preuve est la suivante. Considérons L un langage quelconque de **NP**, M une machine de Turing polynomiale non déterministe qui décide L, et p une borne polynomiale du temps de calcul de M. La preuve consiste à construire une machine de Turing déterministe à partir de M et p, qui sur toute chaîne $x \in \{0,1\}^*$, renvoie une formule SAT f telle que toute instanciation satisfaisant f encode un calcul

acceptant de M sur l'entrée x. Ainsi, f est satisfaisable ssi il existe un calcul acceptant de M sur l'entrée x, c'est-à-dire ssi $x \in L$.

Une fois qu'on a démontré qu'un problème, SAT, est **NP**-difficile, la **NP**-difficulté d'autres problèmes naturels s'obtient « simplement » en exhibant une réduction polynomiale de SAT à ces différents problèmes (voir exercice 1.11). Une des caractéristiques les plus impressionnantes de la théorie de la **NP**-complétude est la facilité[1] avec laquelle on peut trouver ces réductions dans la plupart des cas ; ainsi, avec relativement peu d'efforts on obtient de nombreuses informations cruciales sur le problème traité. Hors mis une poignée de problèmes (importants), la plupart des problèmes naturels de **NP** ont été classés **P** ou **NP**-complets. En fait, il est remarquable que d'autres classes de complexité, en temps ou en mémoire, déterministe ou non, admettent des problèmes naturels complets (selon des réductions adéquates).

La preuve de la **NP**-difficulté de la couverture par sommets repose sur un algorithme polynomial qui, pour toute formule SAT f, renvoie une instance (G, k) telle que G admet une couverture par sommets de taille $\leqslant k$ ssi f est satisfaisable. Il s'ensuit qu'à moins que $\mathbf{P} = \mathbf{NP}$, il n'existe pas d'algorithme polynomial qui distingue les instances positives de la couverture par sommets des instances négatives. Plus précisément, si $\mathbf{P} \neq \mathbf{NP}$, il n'existe pas d'algorithme polynomial qui résout exactement le problème de la couverture par sommets.

Au vu de l'extraordinaire diversité des problèmes **NP**-complets, et comme aucun algorithme polynomial n'a été trouvé pour aucun d'eux en dépit des efforts dépensés pendant de si nombreuses années, la conjecture $\mathbf{P} \neq \mathbf{NP}$ est très largement admise, c'est-à-dire qu'il n'existe aucun algorithme polynomial pour décider un langage **NP**-complet.

La conjecture $\mathbf{P} \neq \mathbf{NP}$ a des conséquences philosophiques profondes. Elle affirme en particulier que trouver une preuve d'une proposition mathématique est qualitativement plus difficile que de vérifier la justesse d'une preuve de cette même proposition, ce que l'on peut reformuler en observant que le langage

$$L = \{(S, 1^n) : \text{la proposition } S \text{ admet une preuve de longueur} \leqslant n\}$$

appartient à **NP**, pour tout système axiomatique raisonnable.

[1] *NDT:* facile, maintenant que l'on dispose d'une base de données importante de problèmes naturels **NP**-difficiles (voir [100]), c'est-à-dire essentiellement depuis l'article fondateur de Karp [172] qui a révélé l'utilité de la notion de **NP**-complétude (introduite indépendamment par Cook et Levin un an auparavant [54, 194]) en exhibant des réductions polynomiales vers plusieurs problèmes fondamentaux issus des divers domaines de l'informatique.

A.3 Problèmes d'optimisation NP et algorithmes d'approximation

Les problèmes d'optimisation combinatoire consistent à sélectionner une « meilleure » solution d'un ensemble fini. Un *problème Π d'optimisation* **NP** est défini par :

- un ensemble d'*instances valides*, D_Π, reconnaissable en temps polynomial. Nous supposerons que tous les nombres spécifiés dans l'entrée sont rationnels, car notre modèle de calcul ne peut pas manipuler des nombres avec une précision arithmétique infinie. La *taille* $|I|$ d'une instance $I \in D_\Pi$ est le nombre de bits utilisés pour écrire I en supposant que tous les nombres spécifiant l'instance sont écrits en binaire.
- un ensemble de *solutions réalisables* $S_\Pi(I)$ pour toute instance $I \in D_\Pi$. Nous supposons que $S_\Pi(I) \neq \varnothing$, et que toute solution $s \in S_\Pi(I)$ est de longueur polynomiale en $|I|$. Nous supposons de plus qu'il existe un algorithme polynomial qui décide si $s \in S_\Pi(I)$ pour toute paire (I, s).
- une *fonction objectif* obj_Π, calculable en temps polynomial, qui associe un nombre rationnel positif à chaque paire (I, s), où I est une instance et s une solution réalisable de I. La fonction objectif est souvent associée à une grandeur physique, un *coût*, une *longueur*, une *taille*, un *poids*, etc.
- enfin, Π est défini comme un *problème de minimisation* ou de *maximisation*.

La restriction d'un problème de minimisation (maximisation) Π aux instances où les coûts sont la somme de coûts élémentaires unitaires, est la variante de *taille minimum*[2] (*maximum*) du problème Π.

Une *solution optimale* pour une instance d'un problème de minimisation (maximisation) est une solution réalisable qui a la plus petite (grande) valeur objectif. $\mathrm{OPT}_\Pi(I)$ désignera la valeur objectif des solutions optimales d'une instance I, que nous abrégerons OPT quand le contexte le permet.

On peut associer à tout problème d'optimisation **NP** un problème de décision en plaçant une borne sur la valeur d'une solution optimale. L'entrée du problème décision associé à un problème d'optimisation **NP**, Π, est une paire (I, B), où I est une instance de Π et B un nombre rationnel. Si π est un problème de minimisation (maximisation), la réponse est « oui » ssi il existe une solution réalisable pour I de coût $\leqslant B$ ($\geqslant B$). Nous dirons que (I, B) est une instance positive dans ce cas, et négative sinon. Un exemple est le problème de décision associé à la couverture par sommets de taille minimum formulé à la section A.1.

Clairement, un algorithme polynomial pour Π permet de résoudre le problème de décision associé — en comparant le coût calculé d'une solution optimale à la borne B. Inversement, les résultats de complexité du problème de décision se transportent sur Π. En effet, la complexité d'un problème

[2] *Cardinality version of* Π, en anglais.

d'optimisation **NP** est établie en démontrant que son problème de décision est **NP**-difficile. En abusant légèrement des définitions, nous dirons alors que le problème d'optimisation est **NP**-difficile.

Un algorithme d'approximation est un algorithme qui produit efficacement une solution réalisable dont la valeur est « proche » de l'optimal. La définition formelle est différente suivant si le problème est de minimisation ou de maximisation. Étant donné un problème de minimisation (maximisation) Π, et une fonction $\delta : \mathbf{N} \to \mathbf{Q}^+$, telle que $\delta \geqslant 1$ ($\delta \leqslant 1$), un algorithme $\mathcal{A}$ est une *δ-approximation* pour Π si pour toute instance I, $\mathcal{A}$ produit en temps polynomial en $|I|$, une solution réalisable s de I telle que $f_\Pi(I,s) \leqslant \delta(|I|) \cdot \mathrm{OPT}(I)$ ($f_\Pi(I,s) \geqslant \delta(|I|) \cdot \mathrm{OPT}(I)$). Clairement, plus le *facteur d'approximation* δ est proche de 1, meilleur est l'algorithme d'approximation.

Nous relaxerons de temps à autre cette définition en autorisant $\mathcal{A}$ à être randomisé, c'est-à-dire à tirer et utiliser des bits aléatoires uniformes et indépendants. Pour un problème de minimisation, nous dirons que $\mathcal{A}$ est une *δ-approximation randomisée pour Π* si pour toute instance I, $\mathcal{A}$ produit en temps polynomial en $|I|$, une solution réalisable s de I telle que :

$$\mathbf{Pr}[f_\Pi(I,s) \leqslant \delta(|I|) \cdot \mathrm{OPT}(I)] \geqslant \frac{1}{2},$$

où la probabilité est calculée sur les bits aléatoires. La définition pour un problème de maximisation est analogue.

Remarque A.1 δ a été défini comme une fonction de la taille de l'entrée ; cependant, il sera parfois plus pratique de le définir en fonction d'un autre paramètre : par exemple, le nombre d'éléments de l'univers pour la couverture par ensembles (voir chapitre 2).

A.3.1 Réductions isofacteur

En général, les réductions polynomiales envoient les solutions optimales sur des solutions optimales, mais les solutions presque-optimales n'importe où, c'est-à-dire sans garantie. En effet, la complexité de l'ensemble des problèmes **NP**-complets est identique du point de vue des solutions exactes (optimales), mais exhibe une grande diversité du point de vue des solutions approchées, évoquée précédemment.

Dans ce livre, nous avons rencontré des problèmes qui semblent différents au premier coup d'œil, mais dont les propriétés d'approximabilité sont liées (par exemple, voir exercice 19.13). Définissons formellement une notion de réduction qui permet ce type de connexions. Plusieurs notions de réductions qui préservent l'approximabilité par une constante ont été définies. La notion suivante est plus stricte car elle conserve la constante elle-même. De tels problèmes sont nécessairement tous deux des problèmes de minimisation ou tous deux de maximisation.

Considérons deux problèmes de minimisation Π_1 et Π_2 (la définition est similaire pour deux problèmes de maximisation). Une *réduction isofacteur*[3] de Π_1 à Π_2 est définie par deux algorithmes polynomiaux f et g tels que :

- pour toute instance I_1 de Π_1, $I_2 = f(I_1)$ est une instance de Π_2 telle que $\mathrm{OPT}_{\Pi_2}(I_2) \leqslant \mathrm{OPT}_{\Pi_1}(I_1)$, et
- pour toute solution t de I_2, $s = g(I_1, t)$ est une solution de I_1 telle que

$$\mathrm{obj}_{\Pi_1}(I_1, s) \leqslant \mathrm{obj}_{\Pi_2}(I_2, t).$$

Clairement, avec une telle réduction, s'il existe une α-approximation pour Π_2, on en déduit une α-approximation pour Π_1 (voir exercice 1.16).

A.4 Classes de complexité randomisées

Certains langages **NP** admettent des certificats positifs en abondance. Cette caractéristique[4] permet de les résoudre efficacement moyennant principalement une source de bits aléatoires. Ces langages appartiennent à la classe **RP**, pour *Temps Polynomial Randomisé*.[5] Un langage $L \in \mathbf{RP}$ s'il existe un polynôme p et une machine de Turing M en temps polynomial telle que pour toute chaîne $x \in \{0,1\}^*$:

- si $x \in L$, alors $M(x, y)$ accepte au moins la moitié des chaînes y de longueur $p(|x|)$, et
- si $x \notin L$, alors pour toute chaîne y de longueur $p(|x|)$, $M(x, y)$ rejette.

Clairement, $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$. Soient $L \in \mathbf{RP}$ et une entrée x. Sélectionnons une chaîne aléatoire y de longueur $p(|x|)$ et exécutons $M(x, y)$. Clairement, le temps de calcul total est polynomial. Il est possible qu'on rejette x par erreur, même si $x \in L$. Cependant, la probabilité d'un tel événement, appelée *probabilité d'erreur*, est inférieure à $1/2$. Remarquons qu'en utilisant l'astuce classique, c'est-à-dire en répétant l'exécution sur différentes chaînes aléatoires indépendantes, on peut réduire la probabilité d'erreur exponentiellement en le nombre d'exécutions.

Un langage L appartient à la classe co-**RP** ssi $\overline{L} \in \mathbf{RP}$. Un tel langage présente des certificats négatifs en abondance. La machine de Turing associée ne peut se tromper que sur les entrées $x \notin L$. Enfin, la classe **ZPP**, pour *temps polynomial probabiliste sans erreur*,[6] des langages qui sont reconnus par une machine de Turing randomisée (c'est-à-dire une machine de Turing ayant accès à une source de bits aléatoires) qui donne toujours la réponse correcte et dont l'*espérance* du temps de calcul est polynomiale. Il est facile de prouver que (voir exercice 1.17) :

$$L \in \mathbf{ZPP} \text{ ssi } L \in (\mathbf{RP} \cap \text{co-}\mathbf{RP}).$$

[3] *Approximation factor preserving reduction*, en anglais.
[4] Les définitions de cette section sont utilisées au chapitre 29.
[5] *Randomized polynomial time class*, en anglais.
[6] *Zero-error probabilistic polynomial time class*, en anglais.

Nous noterons $\mathbf{DTIME}(t)$ la classe des problèmes qui admettent un algorithme déterministe en temps $O(t)$. Par exemple, $\mathbf{P} = \mathbf{DTIME}(\text{poly}(n))$, où $\text{poly}(n) = \bigcup_{k \geqslant 0} n^k$. Nous noterons $\mathbf{ZTIME}(t)$ la classe des problèmes qui admettent un algorithme randomisé dont l'espérance du temps de calcul est $O(t)$. Ainsi, $\mathbf{ZPP} = \mathbf{ZTIME}(\text{poly}(n))$.

A.5 Auto-réductibilité

La plupart des problèmes **NP** connus présentent une propriété intéressante, l'autoréductibilité, qui permet de trouver en temps polynomial une solution (un certificat positif), moyennant un oracle pour le problème de décision associé. Une version légèrement plus élaborée de cette propriété permet de construire un algorithme polynomial exact pour un problème d'optimisation **NP**, moyennant encore une fois un oracle pour le problème de décision associé. Ceci démontre que les principales difficultés de **NP** et des problèmes d'optimisation **NP** sont concentrées sur les problèmes de décisions (voir section 16.2 et exercice 28.7 pour d'autres applications fondamentales de l'autoréductibilité).

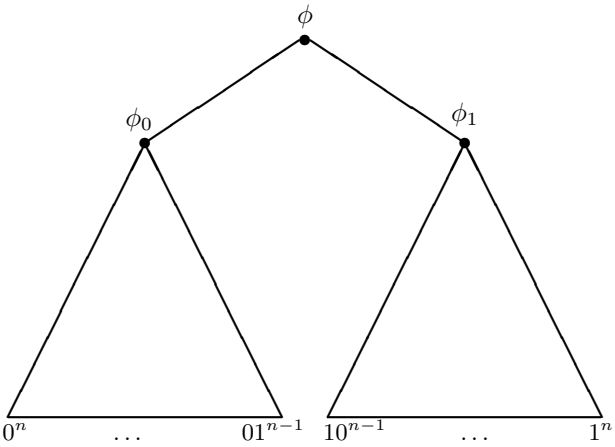
Le cadre le plus simple pour décrire l'autoréductibilité est sans doute SAT. Soit ϕ une formule SAT sur n variables booléennes $x_1, \ldots, x_n$. Nous représenterons une instanciation des n variables par un vecteur de n bits (Vrai = 1 et Faux = 0). Notons S l'ensemble des instanciations satisfaisant ϕ, c'est-à-dire l'ensemble des solutions. Le point important est que si on fixe x_1 à 0 (1), on peut trouver, en temps polynomial, une formule ϕ_0 (ϕ_1) sur les $n-1$ autres variables dont l'ensemble des solutions S_0 (S_1) est précisément l'ensemble des solutions de ϕ où $x_1 = 0$ ($x_1 = 1$).

Exemple A.2 Prenons $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_4)$. Alors $\phi_0 = (x_2 \vee x_3)$ et $\phi_1 = (x_2 \vee x_4)$. □

Cette propriété permet de trouver une solution pour toute formule satisfaisable ϕ à l'aide d'un oracle pour le problème de décision associé à SAT. Déterminons tout d'abord si ϕ_0 est satisfaisable ou non. Si tel est le cas, posons $x_1 = 0$, et recherchons récursivement une solution de ϕ_0. Sinon, posons $x_1 = 1$ (dans ce cas, ϕ_1 est nécessairement satisfaisable), et recherchons récursivement une solution de ϕ_1. Nous pouvons donc réduire l'instance à une instance plus petite et nous obtenons la solution au bout de n itérations.

La représentation arborescente suivante est très pratique. Considérons T un arbre binaire de profondeur n dont les arêtes issues d'un nœud sont étiquetées par le rang du fils correspondant (0 ou 1). On associe à chaque feuille de T l'instanciation des n variables de ϕ représentée par la chaîne de n bits décrivant le chemin depuis la racine. Les feuilles correspondant à des solutions de ϕ sont dites spéciales. On étiquette la racine de T par ϕ, et tous les nœuds internes par les formules dont les solutions sont en bijection avec

les feuilles spéciales contenues dans le sous-arbre correspondant. Ainsi, le « 0-ième » fils de la racine est étiqueté par ϕ_0 et le « 1-ième » par ϕ_1. L'arbre T est l'*arbre d'autoréductibilité* de l'instance ϕ.



Formalisons la notion d'autoréductibilité pour des problèmes d'optimisation **NP**. La formalisation de cette notion pour les problèmes **NP** est plus facile et différée à l'exercice 1.15.

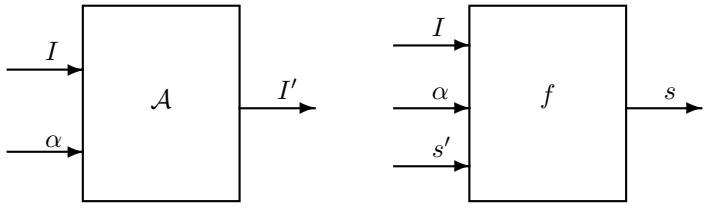
Commençons par illustrer l'autoréductibilité avec la couverture par sommets de taille minimum. Tout d'abord, un oracle pour le problème de décision associé permet de trouver, par dichotomie sur k, la taille $\mathrm{OPT}(G)$ d'une couverture optimale. Pour calculer une couverture optimale, construisons le graphe G' à partir de G en ôtant un sommet v et ses arêtes incidentes, puis calculons $\mathrm{OPT}(G')$. Clairement, v appartient à une couverture optimale ssi $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$. De plus, si v appartient à une couverture optimale, alors l'ajout de v à toute couverture optimale de G' donne une couverture optimale de G. Sinon, toute couverture optimale de G contient nécessairement l'ensemble $N(v)$ des voisins de v (pour couvrir toutes les arêtes incidentes à v). Soit G'' le graphe obtenu à partir de G en ôtant v et $N(v)$. Alors, toute couverture optimale de G'', complétée par $N(v)$, est une couverture optimale de G. Ainsi, dans les deux cas, il s'agit de trouver une couverture optimale dans un graphe plus petit, G' ou G''. Nous pouvons donc bien construire récursivement une couverture optimale de G en temps polynomial.

La réduction ci-dessus du problème de la couverture par sommets à son problème de décision associé fonctionne car nous pouvons démontrer qu'il existe des algorithmes polynomiaux pour :

- construire les graphes plus petits, G' et G'' ;
- calculer la taille optimale d'une couverture de G, consistante avec le choix atomique fait ;
- construire une couverture optimale de G, à partir d'une couverture optimale de l'instance plus petite.

L'autoréductibilité se manifeste de façon différente dans chaque problème. Nous proposons ci-dessous une définition assez générale qui couvre un grand nombre de cas. Afin de faire mieux apparaître l'idée principale sous-jacente, nous préférons en donner ici une définition intuitive, que le lecteur pourra très facilement formaliser.

Nous supposons que les solutions d'une instance du problème d'optimisation **NP** considéré Π ont une certaine *granularité*, c'est-à-dire sont formées de petits morceaux, les *atomes*, qui ont un sens pour le problème. Par exemple, pour la couverture par sommets, chaque atome détermine si un sommet donné appartient ou non à la couverture. Dans ce cas, chaque atome s'écrit avec $O(\log n)$ bits. En fait, tous les problèmes étudiés dans ce livre ont également pour granularité $O(\log n)$. Considérons que c'est aussi le cas pour Π.



Nous dirons que le problème Π est *autoréductible* s'il existe un algorithme polynomial $\mathcal{A}$ et des fonctions calculables en temps polynomial, $f(\cdot,\cdot,\cdot)$ et $g(\cdot,\cdot,\cdot)$, tels que :

- étant donné une instance I et un atome α d'une solution de I, $\mathcal{A}$ renvoie une instance I_α, avec $|I_\alpha| < |I|$. De plus, si on note $S(I \mid \alpha)$ l'ensemble des solutions réalisables de I consistantes avec l'atome α, l'ensemble des solutions réalisables $S(I_\alpha)$ de I_α doit être en bijection avec $S(I \mid \alpha)$. De plus, cette bijection est donnée par la fonction calculable en temps polynomiale $f(\cdot,\cdot,\cdot)$:

$$f(I,\alpha,\cdot) : S(I_\alpha) \to S(I \mid \alpha).$$

- la bijection $f(I,\alpha,\cdot)$ préserve l'ordre des valeurs objectif des solutions. Ainsi, si s'_1 et s'_2 sont deux solutions réalisables de I_α telles que $\mathrm{obj}_\Pi(I_\alpha, s'_1) \leqslant \mathrm{obj}_\Pi(I_\alpha, s'_2)$, et $f(I,\alpha,s'_1) = s_1$ et $f(I,\alpha,s'_2) = s_2$, alors $\mathrm{obj}_\Pi(I, s_1) \leqslant \mathrm{obj}_\Pi(I, s_2)$.
- si on suppose connu le coût d'une solution de I_α, alors le coût optimal d'une solution de $S(I \mid \alpha)$ se calcule en temps polynomial, c'est-à-dire est donné par $g(I,\alpha,\mathrm{OPT}(I_\alpha))$.

Théorème A.3 *Soit Π un problème d'optimisation* **NP** *autoréductible. Il existe un algorithme qui, étant donné un oracle $\mathcal{O}$ pour le problème de décision associé, résout Π (exactement) en temps polynomial.*

Preuve : Comme évoqué précédemment, une recherche dichotomique utilisant l'oracle $\mathcal{O}$ permet de calculer le coût d'une solution optimale en temps polynomial.

Voici comment en déduire un algorithme polynomial $\mathcal{R}$ qui résout Π exactement. Soient $\mathcal{A}$, f, et g les fonctions associées par autoréductibilité à Π, et I une instance de Π. $\mathcal{R}$ commence par trouver un atome d'une solution optimale de I. Un tel atome β est caractérisé par la condition $g(I, \beta, \mathrm{OPT}(I_\beta)) = \mathrm{OPT}(I)$, où $I_\beta = \mathcal{A}(I, \beta)$. Comme les atomes sont codés sur $O(\log n)$ bits, un temps polynomial suffit pour trouver un tel atome par recherche exhaustive. Notons α l'atome trouvé et posons $I_\alpha = \mathcal{A}(I, \alpha)$. $\mathcal{R}$ procède alors récursivement pour trouver une solution optimale s' de I_α. Enfin, il renvoie $f(I, \alpha, s')$ qui est par construction une solution optimale de I. Comme $|I_\alpha| < |I|$, la récursion termine bien en temps polynomial. □

Remarque A.4 Le nombre de chaînes de longueur $O(\log n)$ que l'algorithme $\mathcal{R}$ doit tester pour trouver un bon atome, dépend spécifiquement du problème. Par exemple, pour la couverture par sommet, nous avons sélectionné un sommet v arbitraire et considéré uniquement deux atomes : v appartient ou non à la couverture.

A.6 Notes

La définition d'un problème d'optimisation **NP** est due à Krentel [186]. Les réductions isofacteur sont une version plus contrainte des L-réductions introduites par Papadimitriou et Yannakakis [227]. L'autoréductibilité a été définie par Schnorr [243]. Référez-vous à Khuller et Vazirani [179] pour un exemple de problème non autoréductible, à moins que $\mathbf{P} = \mathbf{NP}$. Reportez-vous aux livres de Garey et Johnson [100] et de Papadimitriou [225], pour plus d'informations sur la **NP**-complétude et la théorie de la complexité.

B Éléments de théorie des probabilités

Nous rappelons ici des faits utiles de la théorie des probabilités. Nous considérons que le lecteur est déjà familier avec ces notions (reportez-vous à la section B.4 pour des références).

B.1 Espérance et moments

Deux quantités fournissent beaucoup d'informations sur une variable aléatoire : sa moyenne, appelée espérance, et sa variance.[1] Une propriété clé de l'espérance, qui simplifie très souvent son calcul, est appelée la *linéarité de l'espérance*. Elle affirme que si $X, X_1, \ldots, X_n$ sont des variables aléatoires telles que $X = c_1 X_1 + \ldots + c_n X_n$, où $c_1, \ldots, c_n$ sont des constantes, alors :

$$\mathbf{E}[X] = c_1 \mathbf{E}[X_1] + \ldots + c_n \mathbf{E}[X_n].$$

En particulier, l'espérance de la somme de variables aléatoires est la somme de leurs espérances. L'utilité de cette propriété vient du fait qu'aucune hypothèse d'indépendance n'est faite sur les variables aléatoires $X_1, \ldots, X_n$. Très souvent, une variable aléatoire compliquée peut se réécrire comme une somme de variables indicatrices (c'est-à-dire de variables aléatoires à valeurs dans $\{0, 1\}$), ce qui simplifie l'évaluation de son espérance.

La variance d'une variable aléatoire X mesure son étalement autour de sa moyenne. Elle est définie par :

$$\boldsymbol{V}[X] = \mathbf{E}[(X - \mathbf{E}[X])^2] = \mathbf{E}[X^2] - \mathbf{E}[X]^2.$$

Sa racine carrée positive est appelée l'*écart type*.[2] La moyenne et l'écart type de X sont notées respectivement $\mu(X)$ et $\sigma(X)$.

Pour tout $k \in \mathbf{N}$, le k-ième *moment* et le k-ième *moment centré* de X sont définis par $\mathbf{E}[X^k]$ et $\mathbf{E}[(X - \mathbf{E}[X])^k]$, respectivement. La variance est donc le deuxième moment centré.

[1] *Expectation and variance*, en anglais.
[2] *Standard deviation*, en anglais.

En général, l'espérance du produit de deux variables aléatoires n'est pas le produit des espérances. Une exception importante est le cas de variables indépendantes. En effet, si X et Y sont deux variables aléatoires indépendantes, alors $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$. Une conséquence directe est que la variance de la somme de deux variables indépendantes est la somme de leurs variances, c'est-à-dire pour toutes variables indépendantes X et Y, $\boldsymbol{V}[X+Y] = \boldsymbol{V}[X] + \boldsymbol{V}[Y]$.

B.2 Déviations de la moyenne

Si X est une variable aléatoire positive dont on connaît l'espérance, alors l'*inégalité de Markov* permet de borner la probabilité de s'écarter de la moyenne : pour tout $t \in \mathbf{R}^+$,

$$\mathbf{Pr}[X \geqslant t] \leqslant \frac{\mathbf{E}[X]}{t}.$$

Il est étonnant de constater que cette inégalité évidente a de très nombreuses applications. Par exemple, elle permet d'obtenir des encadrements avec forte probabilité à partir d'une borne sur l'espérance (par exemple, voir section 14.2).

Si la variance d'une variable aléatoire est faible, alors de grandes déviations de la moyenne sont peu probables. Cette intuition se formalise avec l'*inégalité de Tchebycheff* qui affirme que pour toute variable aléatoire X et tout $a \in \mathbf{R}+$,

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geqslant a] \leqslant \left(\frac{\sigma(X)}{a}\right)^2.$$

Reportez-vous au lemme 28.5 pour un exemple d'application.

Les *tirages de Poisson* sont des tirages itérés indépendants, où chaque tirage a deux résultats possibles, succès ou échec. En général, la probabilité de succès peut varier suivant les tirages. On parle de *tirages de Bernoulli* si la probabilité de succès reste constante d'un tirage à l'autre.

Les inégalités de Chernoff, qui bornent la queue des lois de probabilité des tirages de Poisson, sont très utiles pour analyser des algorithmes. Notons $X_1, \ldots, X_n$ les variables aléatoires associées aux résultats de n tirages de Poisson, où 1 représente un succès et 0 un échec. Posons $\mathbf{Pr}[X_i = 1] = p_i$, avec $0 < p_i < 1$ pour $1 \leqslant i \leqslant n$. Considérons la variable aléatoire $X = X_1 + \ldots + X_n$ et posons $\mu = \mathbf{E}[X] = \sum_{i=1}^{n} p_i$. Alors, à gauche de la moyenne, pour tout $0 < \delta \leqslant 1$,

$$\mathbf{Pr}[X < (1-\delta)\mu] < e^{(-\mu\delta^2/2)}.$$

La borne à droite de la moyenne est plus sophistiquée : pour tout $\delta > 0$,

$$\mathbf{Pr}[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}.$$

Cette expression se simplifie en considérant deux intervalles pour δ. Pour $\delta > 2e - 1$,

$$\mathbf{Pr}[X > (1+\delta)\mu] < 2^{-(1+\delta)\mu},$$

et pour $\delta \leqslant 2e - 1$,

$$\mathbf{Pr}[X > (1+\delta)\mu] < e^{-\mu\delta^2/4}.$$

B.3 Lois de probabilités classiques

Nous présentons ici trois lois de probabilités d'une grande universalité. La loi de probabilités du nombre de succès dans des tirages de Bernoulli est appelée la *loi de probabilité binomiale*. Considérons n tirages de Bernoulli indépendants ayant une probabilité de succès p. La probabilité qu'exactement k tirages soient un succès, pour $0 \leqslant k \leqslant n$, est donnée par :

$$B(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}.$$

La *loi de Poisson* de paramètre $\lambda > 0$ est définie comme suit. Pour tout entier k positif, la probabilité qu'exactement k succès se produisent est :

$$p(k; \lambda) = e^{-\lambda} \frac{\lambda^k}{k!}.$$

La loi de Poisson $p(k; \lambda)$ est la loi limite de la loi binomiale $B(k; n, p)$ lorsque $n \to \infty$ et $np \to \lambda$, pour λ constant. En fait, on rencontre dans de nombreuses applications des tirages de Bernoulli où n est grand, p est petit, et où leur produit $\lambda = np$ est de taille moyenne. Dans ces situations, $p(k; np)$ est une bonne approximation de $B(k; n, p)$.

La *fonction de densité normale* de moyenne μ et d'écart type σ est :

$$n(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

et la *loi normale* est son intégrale,

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \mathrm{d}y.$$

La loi normale est aussi une approximation de la loi binomiale. Établissons ceci pour le cas $p = 1/2$. Soit $n = 2\nu$ un entier pair. Pour $-\nu \leqslant k \leqslant \nu$, posons

$$a_k = a_{-k} = B(\nu + k; 2\nu, 1/2).$$

À la limite quand $\nu \to \infty$ et k varie dans l'intervalle $0 < k < \sqrt{\nu}$, a_k peut être approché par $h\, n(k\, h)$, avec $h = \sqrt{\frac{2}{\nu}} = \frac{2}{\sqrt{n}}$.

B.4 Notes

Pour plus d'informations, référez-vous aux livres de Feller [92], Motwani et Raghavan [218], Spencer [252] et d'Alon et Spencer [8].

Bibliographie

1. A. Agrawal, P. Klein, and R. Ravi. When trees collide : an approximation algorithm for the generalized Steiner network problem on networks. *SIAM Journal on Computing*, 24 :440–456, 1995. (Cité pp. 146, 237)
2. M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2) :781–793, 2004. (Cité p. 386)
3. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, New Jersey, 1993. (Cité p. 119)
4. M. Ajtai. The shortest vector problem in ℓ_2 is NP-hard for randomized reductions. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 10–19, 1998. (Cité p. 325)
5. D. Aldous. The random walk construction for spanning trees and uniform labeled trees. *SIAM Journal on Discrete Mathematics*, 3 :450–465, 1990. (Cité p. 377)
6. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5 :13–51, 1995. (Cité p. 301)
7. N. Alon, A. Frieze, and D. Welsh. Polynomial time randomised approximation schemes for Tutte-Grothendieck invariants : the dense case. *Random Structures and Algorithms*, 6 :459–478, 1995. (Cité p. 380)
8. N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley & Sons, New York, NY, 2000. (Cité pp. 155, 398)
9. J. D. Annan. The complexities of the coefficients of the Tutte polynomial. *Discrete Applied Mathematics*, 57 :93–103, 1995. (Cité p. 380)
10. J.D. Annan. A randomized approximation algorithm for counting the number of forests in dense graphs. *Combinatorics, Probability and Computing*, 3 :273–283, 1994. (Cité p. 377)
11. S. Arora. Polynomial time approximation scheme for Euclidean TSP and other geometric problems. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 2–11, 1996. (Cité p. 99)
12. S. Arora. Nearly linear time approximation scheme for Euclidean TSP and other geometric problems. In *Proc. 38th IEEE Annual Symposium on Foundations of Computer Science*, pages 554–563, 1997. (Cité p. 99)
13. S. Arora and C. Lund. Hardness of approximations. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publishing, Boston, MA, 1997. (Cité p. 369)

14. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 13–22, 1992. (Cité pp. 368, 369)

15. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 106–113, 1998. (Cité p. 99)

16. S. Arora and S. Safra. Probabilistic checking of proofs : a new characterization of NP. In *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 2–13, 1992. (Cité pp. 368, 369)

17. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *Proc. 33rd ACM Symposium on the Theory of Computing*, 2001. (Cité pp. 284, 285)

18. Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithms. *SIAM Journal on Computing*, 27 :291–301, 1998. (Cité p. 220)

19. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, Berlin, 1999. (Cité pp. 11, 369)

20. L. Babai. Trading group theory for randomness. In *Proc. 17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985. (Cité p. 368)

21. V. Bafna, P. Berman, and T. Fujito. Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs. In *Algorithms and Computation, 6th International Symposium, ISAAC*, volume 1004 of *Lecture Notes in Computer Science*, pages 142–151. Springer-Verlag, Berlin, 1995. (Cité p. 66)

22. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2 :198–203, 1981. (Cité p. 146)

23. Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 184–193, 1996. (Cité p. 285)

24. C. Bazgan, M. Santha, and Z. Tuza. Efficient approximation algorithms for the subset-sum problem. In *Proc. 25th International Colloquium on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 387–396. Springer-Verlag, Berlin, 1998. (Cité p. 80)

25. A. Becker and D. Geiger. Approximation algorithms for the loop cutset problem. In *Proc. 10th Conference on Uncertainty in Artificial Intelligence*, pages 60–68, 1994. (Cité p. 66)

26. M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs : How to remove intractability. In *Proc. 20th ACM Symposium on the Theory of Computing*, pages 113–131, 1988. (Cité p. 369)

27. M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32 :171–176, 1989. (Cité p. 369)

28. S.N. Bhatt and F.T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28 :300–343, 1984. (Cité p. 220)

29. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstring. *Journal of the ACM*, 41 :630–647, 1994. (Cité p. 73)

30. M. Blum and S. Kannan. Designing programs that check their work. In *Proc. 21st ACM Symposium on the Theory of Computing*, pages 86–97, 1989. (Cité p. 368)

31. M. Blum, M. Luby, and R. Rubinfeld. Testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47 :549–595, 1993. (Cité p. 368)

32. R. Boppana and M.M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32 :180–196, 1992. (Cité p. 369)

33. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, UK, 1998. (Cité p. 85)

34. J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert spaces. *Israeli J. Math.*, 52 :46–52, 1985. (Cité p. 220)

35. A.Z. Broder. How hard is it to marry at random ? In *Proc. 18th ACM Symposium on the Theory of Computing*, pages 50–58, 1986. (Cité p. 338)

36. A.Z. Broder. Generating random spanning trees. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, pages 442–447, 1989. (Cité p. 377)

37. R. Bubley and M. Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201 :81–88, 1999. (Cité p. 378)

38. G. Calinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 48–52, 1998. (Cité p. 185)

39. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner tree problems. In *Proc. 9th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 192–200, 1998. (Cité p. 375)

40. M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. 40th IEEE Annual Symposium on Foundations of Computer Science*, pages 378–388, 1999. (Cité p. 286)

41. M. Charikar, S. Guha, É. Tardos, and D.B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proc. 31st ACM Symposium on the Theory of Computing*, pages 1–10, 1999. (Cité p. 285)

42. M. Charikar, S. Khuller, D.M. Mount, and G. Narshimhan. Algorithms for facility location problems with outliers. In *Proc. 12th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 642–651, 2001. (Cité p. 270)

43. M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai, and A. Tomkins. Minimizing wirelength in zero and bounded skew clock trees. In *Proc. 10th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 177–184, 1999. (Cité p. 40)

44. C. Chekuri, A. Gupta, and A. Kumar. On the bidirected relaxation for the multiway cut problem. Manuscript, 2002. (Cité p. 381)

45. J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 292–301, 1996. (Cité pp. 254, 255, 259)

46. B. Chor and M. Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11 :511–523, 1998. (Cité p. 299)

47. E.-A. Choukhmane. Une heuristique pour le problème de l'arbre de Steiner. *RAIRO Rech. Opér.*, 12 :207–212, 1978. (Cité p. 40)

48. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976. (Cité p. 40)

49. F. Chudak, M.X. Goemans, D. Hochbaum, and D.P. Williamson. A primal–dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters*, 22 :111–118, 1998. (Cité pp. 66, 146)

50. F. Chudak, T. Roughgarden, and D.P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal–dual method and Lagrangian relaxation. Manuscript, 2000. (Cité p. 282)

51. V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4 :233–235, 1979. (Cité pp. 27, 131)

52. V. Chvátal. *Linear Programming*. W.H. Freeman and Co., New York, NY, 1983. (Cité p. 119)

53. E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin backing : a survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publishing, Boston, MA, 1997. (Cité p. 85)

54. S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971. (Cité pp. 11, 387)

55. W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, New York, NY, 1998. (Cité p. 119)

56. C. Cooper and A. Frieze. Mixing properties of the Swendsen-Wang process on classes of graphs. *Random Structures Algorithms*, 15 :242–261, 1999. (Cité p. 380)

57. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Second edition. MIT Press and McGraw-Hill, 2001. (Cité p. 11)

58. R. Courant and H. Robbins. *What Is Mathematics ?* Oxford University Press, New York, NY, 1941. (Cité p. 40)

59. M. Cryan, M. Dyer, L. Goldberg, M. Jerrum, and R. Martin. A polynomial-time alogarithm to approximately count contingency tables when the number of rows is constant. In *Proc. 43rd IEEE Annual Symposium on Foundations of Computer Science*, 2002. (Cité p. 381)

60. P. Dagum, M. Luby, M. Mihail, and U.V. Vazirani. Polytopes, permanents and graphs with large factors. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 412–421, 1988. (Cité p. 377)

61. E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23 :864–894, 1994. (Cité p. 50)

62. G.B. Dantzig. *Linear Programming and Extensions*. Reprint of the 1968 corrected edition. Princeton University Press, Princeton, NJ, 1998. (Cité p. 119)

63. G.B. Dantzig, L.R. Ford, and D.R. Fulkerson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954. (Cité p. 259)

64. G.B. Dantzig, L.R. Ford, and D.R. Fulkerson. A primal–dual algorithm for linear programs. In H.W. Kuhn and A.W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, Princeton, NJ, 1956. (Cité p. 146)

65. I. Dinur, V. Guruswami, S. Khot, and O. Regev. The hardness of hypergraph vertex cover. Manuscript, 2002. (Cité p. 381)

66. G. Dobson. Worst-case analysis of greedy heuristics for integer programming with non-negative data. *Mathematics of Operations Research*, 7 :515–531, 1982. (Cité p. 131)

67. P. Drineas, R. Kannan, A. Frieze, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *Proc. 10th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 291–299, 1999. (Cité p. 286)

68. D.Z. Du and F.K. Hwang. Gilbert-Pollack conjecture on Steiner ratio is true. *Algorithmica*, 7 :121–135, 1992. (Cité p. 40)

69. A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. In M. Nielsen and B. Rovan, editors, *Proceedings 25th International Symposium on Mathematical Foundations of Computer Science (MFCS 2000), Bratislava (Slovakia)*, volume 1893 of *Lecture Notes in Computer Science*, pages 323–332. Springer-Verlag, 2000. (Cité p. 338)

70. M. Dyer. Approximate counting by dynamic programming. Manuscript, 2002. (Cité p. 381)

71. M. Dyer, R. Kannan, and J. Mount. Sampling contingency tables. *Random Structures and Algorithms*, 10 :487–506, 1997. (Cité p. 378)

72. M.E. Dyer, A. Frieze, and M.R. Jerrum. Approximately counting hamilton cycles in dense graphs. *SIAM Journal on Computing*, 27 :1262–1272, 1998. (Cité p. 379)

73. M.E. Dyer, A. Frieze, and M.R. Jerrum. On counting independent sets in sparse graphs. In *Proc. 40th IEEE Annual Symposium on Foundations of Computer Science*, pages 210–217, 1999. (Cité p. 379)

74. M.E. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38 :1–17, 1991. (Cité p. 376)

75. J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards. Section B*, 69 :125–130, 1965. (Cité p. 116)

76. J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17 :449–467, 1965. (Cité p. 11)

77. J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards. Section B*, 71 :233–240, 1967. (Cité p. 236)

78. J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1 :127–136, 1971. (Cité p. 116)

79. J. Edmonds. Matroid intersection. *Annals of Discrete Mathematics*, 4 :185–204, 1979. (Cité p. 255)

80. P. Erdős. Gráfok páros körüljárású részgráfjairól (On bipartite subgraphs of graphs, in Hungarian). *Mat. Lapok*, 18 :283–288, 1967. (Cité p. 11)

81. P. Erdős and J.L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14 :298–301, 1973. (Cité p. 155)

82. G. Even, J. Naor, B. Schieber, and S. Rao. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM*, 47 :585–616, 2000. (Cité p. 198)

83. G. Even, J. Naor, B. Schieber, and L. Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. In *Proc. 4th Israel Symposium on Theory of Computing and Systems*, pages 78–88, 1996. (Cité p. 186)

84. G. Even, J. Naor, and L. Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 310–319, 1996. (Cité p. 186)

85. T. Feder and M. Mihail. Balanced matroids. In *Proc. 24th ACM Symposium on the Theory of Computing*, pages 26–38, 1992. (Cité p. 377)

86. U. Feige. Approximating the bandwidth via volume respecting embeddings. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 90–99, 1998. (Cité p. 218)

87. U. Feige. A treshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45 :634–652, 1998. (Cité pp. 27, 367, 369)

88. U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX-CUT and MAX DICUT. In *Proc. 3rd Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995. (Cité p. 301)

89. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, pages 2–12, 1991. (Cité pp. 368, 369)

90. U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *Proc. 41st IEEE Annual Symposium on Foundations of Computer Science*, pages 105–115, 2000. (Cité pp. 220, 374)

91. U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for MAX-CUT. In *Proc. 33rd ACM Symposium on the Theory of Computing*, 2001. (Cité p. 301)

92. W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, New York, NY, 1950. (Cité p. 398)

93. W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1 :349–355, 1981. (Cité p. 85)

94. A. Freund and H. Karloff. A lower bound of $8/(7+\frac{1}{k-1})$ on the integrality ratio of the Calinescu–Karloff–Rabani relaxation for multiway cut. *Information Processing Letters*, 75 :43–50, 2000. (Cité p. 186)

95. A. Frieze. On the Lagarias–Odlyzko algorithm for the subset sum problem. *SIAM Journal on Computing*, 15 :536–539, 1986. (Cité p. 323)

96. A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12 :23–39, 1982. (Cité p. 37)

97. A. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18 :67–81, 1997. (Cité p. 301)

98. M.R. Garey, R.L. Graham, and J.D. Ullman. An analysis of some packing algorithms. In *Combinatorial Algorithms (Courant Computer Science Symposium, No. 9)*, pages 39–47, 1972. (Cité p. 11)

99. M.R. Garey and D.S. Johnson. Strong NP-completeness results : motivation, examples, and implications. *Journal of the ACM*, 25 :499–508, 1978. (Cité p. 80)

100. M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York, NY, 1979. (Cité pp. 11, 387, 394)

101. N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996. (Cité p. 282)

102. N. Garg, H. Saran, and V.V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. *SIAM Journal on Computing*, 29 :159–179, 1999. (Cité p. 374)

103. N. Garg, V.V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proc. 21st International Colloquium on Automata, Languages, and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 487–498. Springer-Verlag, Berlin, 1994. (Cité p. 186)

104. N. Garg, V.V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25 :235–251, 1996. (Cité p. 199)

105. N. Garg, V.V. Vazirani, and M. Yannakakis. Primal–dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18 :3–20, 1997. (Cité pp. 169, 170, 171)

106. C.F. Gauss. *Disquisitiones Arithmeticae*. English edition translated by A.A. Clarke. Springer-Verlag, New York, NY, 1986. (Cité p. 325)

107. E.N. Gilbert and H.O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16 :1–29, 1968. (Cité p. 40)

108. M.X. Goemans and D.J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60 :145–166, 1993. (Cité p. 256)

109. M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, É. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Proc. 5th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 223–232, 1994. (Cité pp. 253, 259)

110. M.X. Goemans and J. Kleinberg. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, 11 :196–204, 1998. (Cité p. 371)

111. M.X. Goemans and D.P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7 :656–666, 1994. (Cité pp. 154, 155)

112. M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24 :296–317, 1995. (Cité pp. 146, 231, 232, 237)

113. M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42 :1115–1145, 1995. (Cité pp. 299, 301)

114. M.X. Goemans and D.P. Williamson. The primal–dual method for approximation algorithms and its applications to network design problems. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 144–191. PWS Publishing, Boston, MA, 1997. (Cité pp. 146, 237)

115. O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71, 1999. (Cité p. 325)

116. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *SIAM Journal on Computing*, 18 :186–208, 1989. (Cité p. 368)

117. R.E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of the SIAM*, 9 :551–570, 1961. (Cité p. 50)

118. T.F. Gonzalez. Clustering to minimize the maximum inter-cluster distance. *Theoretical Computer Science*, 38 :293–306, 1985. (Cité p. 56)

119. V. Gore and M. Jerrum. The Swendsen-Wang process does not always mix rapidly. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 674–681, 1997. (Cité p. 380)

120. R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45 :1563–1581, 1966. (Cité pp. 11, 91)

121. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17 :416–429, 1969. (Cité p. 91)

122. M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th IEEE Annual Symposium on Foundations of Computer Science*, pages 640–646, 1995. (Cité p. 99)

123. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1 :169–197, 1981. (Cité pp. 119, 189)

124. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Second edition. Springer-Verlag, Berlin, 1993. (Cité p. 119)

125. S. Guha and S. Khuller. Greedy strikes back : Improved facility location algorithms. *Journal of Algorithms*, 31 :228–248, 1999. (Cité p. 272)

126. V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint and related problems. In *Proc. 31st ACM Symposium on the Theory of Computing*, pages 19–28, 1999. (Cité p. 171)

127. D. Gusfield and R. W. Irving. *The Stable Marriage Problem : Structure and Algorithms*. MIT Press, Cambridge, MA, 1989. (Cité p. 379)

128. L.A. Hall. Approximation algorithms for scheduling. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 1–45. PWS Publishing, Boston, MA, 1997. (Cité p. 162)

129. J. Hastad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 627–636, 1996. (Cité p. 369)

130. J. Hastad. Some optimal inapproximability results. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 1–10, 1997. (Cité pp. 367, 368)

131. M. Held and R.M. Karp. The traveling-salesman and minimum cost spanning trees. *Operations Research*, 18 :1138–1162, 1970. (Cité p. 258)

132. D. S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22 :148–162, 1982. (Cité p. 272)

133. D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11 :555–556, 1982. (Cité pp. 26, 139)

134. D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, Boston, MA, 1997. (Cité p. 11)

135. D.S. Hochbaum and D.B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33 :533–550, 1986. (Cité p. 58)

136. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems : theoretical and practical results. *Journal of the ACM*, 34 :144–162, 1987. (Cité p. 91)

137. D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors : using the dual approximation approach. *SIAM Journal on Computing*, 17 :539–551, 1988. (Cité p. 162)

138. J.A. Hoogeveen. Analysis of Christofides' heuristic : some paths are more difficult than cycles. *Operations Research Letters*, 10 :291–295, 1991. (Cité p. 37)

139. E. Horowitz and S.K. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23 :317–327, 1976. (Cité p. 91)

140. W.L. Hsu and G.L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1 :209–216, 1979. (Cité p. 58)

141. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, Netherlands, 1992. (Cité p. 40)

142. O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22 :463–468, 1975. (Cité p. 80)

143. R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proc. 30st IEEE Annual Symposium on Foundations of Computer Science*, pages 248–253, 1989. (Cité p. 369)

144. A. Iwainsky, E. Canuto, O. Taraszow, and A. Villa. Network decomposition for the optimization of connection structures. *Networks*, 16 :205–235, 1986. (Cité p. 40)

145. K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 1 :39–60, 2001. (Cité p. 259)

146. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proc. 34th ACM Symposium on the Theory of Computing*, 2002. (Cité pp. 286, 368)

147. K. Jain, I. I. Măndoiu, V.V. Vazirani, and D. P. Williamson. Primal–dual schema based approximation algorithms for the element connectivity problem. In *Proc. 10th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 484–489, 1999. (Cité p. 374)

148. K. Jain and V.V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. In *Proc. 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 1913 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000. (Cité p. 270)

149. K. Jain and V.V. Vazirani. Approximation algorithms for the metric facility location and k-median problems using the primal–dual schema and Lagrangian relaxation. *Journal of the ACM*, 48 :274–296, 2001. (Cité pp. 272, 284, 285)

150. M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method : an approach to approximate counting. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 482–520. PWS Publishing, Boston, MA, 1997. (Cité p. 339)

151. M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Electronic Colloquium on Computational Complexity*, pages TR00–079, 2000. (Cité pp. 376, 378)

152. M.R. Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms*, 7, 1995. (Cité p. 379)

153. M.R. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18 :1149–1178, 1989. (Cité p. 338)

154. M.R. Jerrum and A. Sinclair. Polynomial time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22 :1087–1116, 1993. (Cité p. 380)

155. M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43 :169–188, 1986. (Cité pp. 336, 337)

156. T. Jiang, M. Li, and D. Du. A note on shortest common superstrings with flipping. *Information Processing Letters*, 44 :195–199, 1992. (Cité p. 73)

157. D.S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, MA, 1973. (Cité p. 84)

158. D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9 :256–278, 1974. (Cité pp. 11, 27, 155)

159. J. Kahn, J.H. Kim, L. Lovász, and V.H. Vu. The cover time, the blanket time, and the Matthews bound. In *Proc. 41st IEEE Annual Symposium on Foundations of Computer Science*, pages 467–475, 2000. (Cité p. 375)

160. M. Kaib and C.-P. Schnorr. The generalized Gauss reduction algorithm. *Journal of Algorithms*, 21(3) :565–578, 1996. (Cité p. 321)

161. R. Kannan. Algorithmic geometry of numbers. In *Annual Review of Computer Science, Vol. 2*, pages 231–267. Annual Reviews, Palo Alto, CA, 1987. (Cité p. 325)

162. R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3) :415–440, 1987. (Cité p. 325)

163. R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $o^*(n^5)$ volume algorithm for convex bodies. *Random Structures and Algorithms*, 11 :1–50, 1997. (Cité p. 376)

164. D. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29 :492–514, 1999. (Cité p. 338)

165. D. Karger, P. Klein, C. Stein, M. Thorup, and N. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 668–678, 1999. (Cité p. 186)

166. D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45 :246–265, 1998. (Cité pp. 299, 301)

167. D. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4) :601–640, 1996. (Cité p. 337)

168. H. Karloff. *Linear Programming*. Birkhäuser, Boston, MA, 1991. (Cité p. 119)

169. H. Karloff. How good is the Goemans-Williamson MAX CUT algorithm. *SIAM Journal on Computing*, 29 :336–350, 1999. (Cité p. 301)

170. H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX-3SAT ? In *Proc. 38th IEEE Annual Symposium on Foundations of Computer Science*, pages 406–415, 1997. (Cité p. 369)

171. N. Karmakar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin packing problem. In *Proc. 23rd IEEE Annual Symposium on Foundations of Computer Science*, pages 312–320, 1982. (Cité p. 85)

172. R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, 1972. (Cité pp. 11, 387)

173. R.M. Karp and M. Luby. Monte Carlo algorithms for enumeration and reliability problems. In *Proc. 24th IEEE Annual Symposium on Foundations of Computer Science*, pages 56–64, 1983. (Cité pp. 336, 338)

174. R.M. Karp, M. Luby, and N. Madras. Monte Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10 :429–448, 1989. (Cité p. 338)

175. A. Karzanov and L. Khachiyan. On the conductance of order Markov chains. Technical Report DCS 268, Rutgers University, 1990. (Cité p. 378)

176. P.W. Kasteleyn. Graph theory and crystal physics. In F. Harary, editor, *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, New York, NY, 1967. (Cité p. 376)

177. S. Khuller, R. Pless, and Y.J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242 :237–245, 2000. (Cité p. 57)

178. S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms*, 21 :434–450, 1996. (Cité p. 374)

179. S. Khuller and V.V. Vazirani. Planar graph colourability is not self-reducible, assuming $P \neq NP$. *Theoretical Computer Science*, 88(1) :183–190, 1991. (Cité p. 394)

180. S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 42, 2 :214–235, 1994. (Cité p. 255)

181. P. Klein, S. Rao, A. Agrawal, and R. Ravi. An approximate max-flow min-cut relation for undirected multicommodity flow, with applications. *Combinatorica*, 15 :187–202, 1995. (Cité pp. 199, 220)

182. D.E. Knuth. *The Art of Computer Programming. Vol. 2. Seminumerical Algorithms.* Second edition. Addison-Wesley, Reading, MA, 1981. (Cité p. 299)

183. A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Math. Annalen*, 6 :366–389, 1873. (Cité p. 323)

184. M. Korupolu, C. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proc. 9th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 1–10, 1998. (Cité p. 284)

185. L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15 :141–145, 1981. (Cité p. 40)

186. M.W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36 :490–509, 1988. (Cité p. 394)

187. H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2 :83–97, 1955. (Cité p. 146)

188. J. Lagarias. Worst case complexity bounds for algorithms in the the theory of integral quadratic forms. *Journal of Algorithms*, 1 :142–186, 1980. (Cité p. 325)

189. J. Lagarias, H.W. Lenstra, Jr., and C.-P. Schnorr. Korkin–Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10 :333–348, 1990. (Cité p. 325)

190. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46 :787–832, 1999. (Cité p. 219)

191. A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261 :513–534, 1982. (Cité p. 325)

192. J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46 :259–271, 1990. (Cité p. 162)

193. H.W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8 :538–548, 1983. (Cité p. 85)

194. L.A. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9 :115–116, 1973. English translation in *Problems of Information Transmission* 9 :265–266. (Cité pp. 11, 387)

195. M. Li. Towards a DNA sequencing theory. In *Proc. 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 125–134, 1990. (Cité p. 27)

196. J. H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44 :245–249, 1992. (Cité p. 282)

197. J. H. Lin and J. S. Vitter. ε-approximation with minimum packing constraint violation. In *Proc. 24th ACM Symposium on the Theory of Computing*, pages 771–782, 1992. (Cité p. 285)

198. N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15 :215–245, 1995. (Cité pp. 218, 220, 298)

199. C.H.C. Little. An extension of Kasteleyn's method of enumerating 1-factors of planar graphs. In D. Holton, editor, *Proc. 2nd Australian Conference on Combinatorial Mathematics*, volume 403 of *Lecture Notes in Computer Science*, pages 63–72. Springer-Verlag, Berlin, 1974. (Cité p. 376)

200. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13 :383–390, 1975. (Cité pp. 11, 27, 131)

201. L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity.* CBMS-NSF Regional Conference Series in Applied Mathematics, 50. SIAM, Philadelphia, PA, 1986. (Cité p. 323)

202. L. Lovász. *Combinatorial Problems and Exercises.* Second edition. North-Holland, Amsterdam–New York, 1993. (Cité pp. 119, 377, 379)

203. L. Lovász and M.D. Plummer. *Matching Theory.* North-Holland, Amsterdam–New York, 1986. (Cité pp. 9, 11, 119)

204. L. Lovász and A. Schrijver. Cones of matrices and set functions, and 0-1 optimization. *SIAM Journal on Optimization*, 1 :166–190, 1990. (Cité p. 301)

205. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8 :261–277, 1988. (Cité p. 369)

206. M. Luby and E. Vigoda. Approximately counting up to four. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 682–687, 1997. (Cité p. 379)

207. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41 :960–981, 1994. (Cité pp. 27, 369)

208. S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algoirthms. In *Proc. 36th IEEE Annual Symposium on Foundations of Computer Science*, pages 162–169, 1995. (Cité p. 301)

209. M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proc. 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 2129 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2001. (Cité pp. 271, 272)

210. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proc. 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 2462 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002. (Cité p. 272)

211. P. Matthews. Generating random linear extensions of a partial order. *The Annals of Probability*, 19 :1367–1392, 1991. (Cité p. 378)

212. L. McShine and P. Tetali. On the mixing time of the triangulation walk and other Catalan structures. *Randomization methods in Algorithm Design, DIMACS-AMS*, 43 :147–160, 1998. (Cité p. 378)

213. D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proc. 39th IEEE Annual Symposium on Foundations of Computer Science*, pages 92–98, 1998. (Cité p. 373)

214. M. Mihail. On coupling and the approximation of the permanent. *Information Processing Letters*, 30 :91–95, 1989. (Cité p. 338)

215. M. Mihail. Set cover with requirements and costs evolving over time. In *International Workshop on Randomization, Approximation and Combinatorial Optimization*, volume 1671 of *Lecture Notes in Computer Science*, pages 63–72. Springer-Verlag, Berlin, 1999. (Cité p. 130)

216. J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions : a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28 :1298–1309, 1999. (Cité p. 99)

217. B. Morris. Improved bounds for sampling contingency tables. In *International Workshop on Randomization, Approximation and Combinatorial Optimization*, volume 1671 of *Lecture Notes in Computer Science*, pages 121–129. Springer-Verlag, Berlin, 1999. (Cité p. 378)

218. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995. (Cité p. 398)

219. J. Naor and L. Zosin. A 2-approximation algorithm for the directed multiway cut problem. In *Proc. 38th IEEE Annual Symposium on Foundations of Computer Science*, pages 548–553, 1997. (Cité p. 186)

220. M. Naor, L. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. 36th IEEE Annual Symposium on Foundations of Computer Science*, pages 182–191, 1995. (Cité p. 369)

221. G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, NY, 1988. (Cité p. 119)

222. G.L. Nemhauser and L.E. Trotter. Vertex packings : structural properties and algorithms. *Mathematical Programming*, 8 :232–248, 1975. (Cité p. 139)

223. Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM, Philadelphia, PA, 1994. (Cité p. 301)

224. M.L. Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM J. on Matrix Analysis and Appl.*, 13 :256–268, 1992. (Cité p. 301)

225. C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994. (Cité p. 394)

226. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982. (Cité pp. 11, 119)

227. C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43 :425–440, 1991. (Cité pp. 368, 394)

228. C.H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18 :1–11, 1993. (Cité p. 37)

229. M. Pinsker. On the complexity of a concentrator. In *Proc. 7th Annual Teletraffic Conference*, pages 318/1–318/4, 1973. (Cité p. 199)

230. J. Plesník. A bound for the Steiner tree problem in graphs. *Math. Slovaca*, 31 :155–163, 1981. (Cité p. 40)

231. V.R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4 :214–220, 1975. (Cité p. 10)

232. H. J. Prömel and A. Steger. RNC-approximation algorithms for the Steiner problem. In *Proc. Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *Lecture Notes in Computer Science*, pages 559–570. Springer-Verlag, Berlin, 1997. (Cité p. 236)

233. M.O. Rabin. Probabilistic algorithms. In J.F. Traub, editor, *Algorithms and Complexity, Recent Results and New Directions*, pages 21–39. Academic Press, New York, NY, 1976. (Cité p. 11)

234. P. Raghavan. Probabilistic construction of deterministic algorithms : approximating packing integer programs. *Journal of Computer and System Sciences*, 37 :130–143, 1988. (Cité p. 155)

235. S. Rajagopalan and V.V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem. In *Proc. 10th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 742–751, 1999. (Cité pp. 235, 372)

236. S. Rajagopalan and V.V. Vazirani. Primal–dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28 :526–541, 1999. (Cité p. 131)

237. D. Randall and D.B. Wilson. Sampling spin configurations of an Ising system. In *Proc. 10th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages S959–960, 1999. (Cité p. 380)

238. S. Rao and W.D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 540–550, 1998. (Cité p. 99)

239. S.K. Rao, P. Sadayappan, F.K. Hwang, and P.W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7 :277–288, 1992. (Cité p. 37)

240. R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27 :763–803, 1998. (Cité p. 369)

241. S.K. Sahni and T.F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23 :555–565, 1976. (Cité p. 40)

242. H. Saran and V.V. Vazirani. Finding k-cuts within twice the optimal. *SIAM Journal on Computing*, 24 :101–108, 1995. (Cité p. 50)

243. C.P. Schnorr. Optimal algorithms for self-reducible problems. In *Proc. 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337, 1976. (Cité p. 394)

244. C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53 :201–224, 1987. (Cité p. 325)

245. P. Schreiber. On the history of the so-called Steiner Weber problem. *Wiss. Z. Ernst-Moritz-Arndt-Univ. Greifswald, Math.-nat.wiss. Reihe, 35*, 3, 1986. (Cité p. 40)

246. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, NY, 1986. (Cité p. 119)

247. P.D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15 :281–288, 1995. (Cité p. 375)

248. D.B. Shmoys, É. Tardos, and K.I. Aardal. Approximation algorithms for facility location problems. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 265–274, 1997. (Cité p. 272)

249. D.B. Shmoys and D.P. Williamson. Analyzing the Held-Karp TSP bound : a monotonicity property with applications. *Information Processing Letters*, 35 :281–285, 1990. (Cité p. 259)

250. A. Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1 :351–370, 1992. (Cité p. 220)

251. A. Sinclair. *Algorithms for Random Generation and Counting : a Markov Chain Approach*. Birkhäuser, Boston, MA, 1993. (Cité p. 339)

252. J. Spencer. *Ten Lectures on the Probabilistic Method*. SIAM, Philadelphia, PA, 1987. (Cité pp. 155, 398)

253. A. Srinivasan. Improved approximations of packing and covering problems. In *Proc. 27th ACM Symposium on the Theory of Computing*, pages 268–276, 1995. (Cité p. 139)

254. R.H. Swendsen and J.S. Wang. Non-universal critical dynamics in Monte Carlo simulations. *Physics Review Letters*, 58 :86–90, 1987. (Cité p. 380)

255. R.E. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, PA, 1983. (Cité p. 11)

256. L. Trevisan. Non-approximability results for optimization problems on bounded degree instance. In *Proc. 33rd ACM Symposium on the Theory of Computing*, 2001. (Cité p. 372)

257. J.D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971. (Cité p. 85)

258. L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8 :189–201, 1979. (Cité p. 338)

259. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38 :49–95, 1996. (Cité p. 301)

260. V.V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computation*, 80 :152–164, 1989. (Cité p. 376)

261. V.V. Vazirani and M. Yannakakis. Suboptimal cuts : their enumeration, weight and number. In *Proc. 19th International Colloquium on Automata, Languages, and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 366–377. Springer-Verlag, Berlin, 1992. (Cité p. 337)

262. D.L. Vertigan and D.J.A. Welsh. The computational complexity of the Tutte plane. *Combinatorics, Probability and Computing*, 1 :181–187, 1992. (Cité p. 380)

263. E. Vigoda. Improved bounds for sampling colorings. In *Proc. 40th IEEE Annual Symposium on Foundations of Computer Science*, pages 51–59, 1999. (Cité p. 379)

264. V.G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3 :25–30, 1964 (in Russian). (Cité p. 11)

265. D.J.A. Welsh. *Knots, Colourings and Counting*. Cambridge University Press, Cambridge, UK, 1993. (Cité p. 380)

266. A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30 :729–735, 1983. (Cité p. 24)

267. D.P. Williamson, M.X. Goemans, M. Mihail, and V.V. Vazirani. A primal–dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15 :435–454, 1995. (Cité pp. 146, 252, 259)

268. D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proc. 30th ACM Symposium on the Theory of Computing*, pages 296–303, 1996. (Cité p. 377)

269. H. Wolkowitz. Some applications of optimization in matrix theory. *Linear Algebra and its Applications*, 40 :101–118, 1981. (Cité p. 301)

270. L.A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Study*, 13 :121–134, 1980. (Cité p. 259)

271. M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 3 :475–502, 1994. (Cité p. 155)

272. A.Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9 :463–470, 1993. (Cité p. 236)

273. A.Z. Zelikovsky and I. I. Măndoiu. Practical approximation algorithms for zero- and bounded-skew trees. In *Proc. 12th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 407–416, 2001. (Cité p. 40)

Index des problèmes

Index

Glossaire des mots anglais

Achevé d'imprimer sur les presses de l'Imprimerie BARNÉOUD
B.P. 44 - 53960 BONCHAMP-LÈS-LAVAL
Dépôt légal : février 2006 - N° d'imprimeur : 602019
Imprimé en France