Sertan Girgin Manuel Loth
Rémi Munos Philippe Preux
Daniil Ryabko (Eds.)

LNAI 5323

Recent Advances in Reinforcement Learning

8th European Workshop, EWRL 2008
Villeneuve d'Ascq, France, June/July 2008
Revised and Selected Papers

Springer

Lecture Notes in Artificial Intelligence 5323

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Sertan Girgin Manuel Loth
Rémi Munos Philippe Preux
Daniil Ryabko (Eds.)

Recent Advances in Reinforcement Learning

8th European Workshop, EWRL 2008
Villeneuve d'Ascq, France, June 30-July 3, 2008
Revised and Selected Papers

Springer

Volume Editors

Sertan Girgin
INRIA Lille-Nord Europe, 59650 Villeneuve d'Ascq, France
E-mail: sertan@ceng.metu.edu.tr

Manuel Loth
Rémi Munos
Philippe Preux
Daniil Ryabko
INRIA, LIFL, CNRS, Université de Lille
Villeneuve d'Ascq, France
E-mail: {manuel.loth,remi.munos,philippe.preux,daniil.ryabko}@inria.fr

Library of Congress Control Number: 2008940489

CR Subject Classification (1998): I.2.6, I.2, F.1, G.3, C.1.3

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN	0302-9743
ISBN-10	3-540-89721-6 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-89721-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12573929 06/3180 5 4 3 2 1 0

Foreword

In the summer of 2008, reinforcement learning researchers from around the world gathered in the north of France for a week of talks and discussions on reinforcement learning, on how it could be made more efficient, applied to a broader range of applications, and utilized at more abstract and symbolic levels. As a participant in this 8th European Workshop on Reinforcement Learning, I was struck by both the quality and quantity of the presentations. There were four full days of short talks, over 50 in all, far more than there have been at any previous meeting on reinforcement learning in Europe, or indeed, anywhere else in the world. There was an air of excitement as substantial progress was reported in many areas including Computer Go, robotics, and fitted methods. Overall, the work reported seemed to me to be an excellent, broad, and representative sample of cutting-edge reinforcement learning research. Some of the best of it is collected and published in this volume.

The workshop and the papers collected here provide evidence that the field of reinforcement learning remains vigorous and varied. It is appropriate to reflect on some of the reasons for this. One is that the field remains focused on a problem — sequential decision making — without prejudice as to solution methods. Another is the existence of a common terminology and body of theory. Reinforcement learning overlaps with many fields (optimal control, artificial intelligence, neuroscience, psychology, economics) that use different terminology for similar concepts; reinforcement learning is part of the long process of creating a theoretical framework common to all these fields. Another strength of reinforcement learning is that its research community continues to embrace diversity, both in its methodologies, empirical and theoretical, and in the time-scale of its ambitions, from near-term applications to the long-term goal of understanding human-level intelligence. Related to this is the continuing influx of new people and ideas from other fields and from around the world. If reinforcement learning can remain open to diverse inspiration while remaining focused on the ubiquitous problems of prediction and control, then its future seems bright indeed.

Richard Sutton

Preface

This volume contains a selection of papers dedicated to the field of reinforcement learning. This volume is an outcome of the 8th European Workshop on Reinforcement Learning (EWRL), that was held in Villeneuve d'Ascq, France, from June 30 to July 3, 2008. As the eighth in the series, the event was a distant follower of the first workshop that was held in Brussels, Belgium, in 1994. Since then, with an average bi-annual frequency, EWRL has gathered mostly European researchers, aiming at being a very open forum dealing with the current research in reinforcement learning. While keeping this openness in the organization, we thought the time has come to make EWRL something bigger; we have tried to gather together researchers world-wide, to have a great scientific event entirely dedicated to the research in reinforcement learning. Still, we wished to keep it wide open to students, PhD students, but also, future PhD students, and let them hear and meet some of the top researchers in the field.

The call for papers attracted 61 propositions, among which were 13 copyrighted papers, and 1 tutorial. Each non-copyrighted paper was reviewed by three reviewers. A light selection for presentation during the workshop resulted in the presentation of 50 papers and the tutorial. Based on these reviews, contributors were invited to improve their papers and resubmit them for publication in the proceedings. After careful reviewing, we selected 21 papers to be included in this volume.

The program of the workshop also included three invited speakers, namely, Richard S. Sutton, from the University of Alberta in Edmonton, Canada, Dimitri Bertsekas, from the Massachussetts Institute of Technology, USA, and Jan Peters, from the Max Planck Institute, Tübingen, Germany.

The workshop itself gathered 105 participants, among them, 44 PhD students, 10 post-doctoral fellows, 39 academics, 8 undergarduate students, and 4 researchers working in private companies. While 47 % of the participants were French, 16 % were Belgian, 9 % were Canadians, 7 % were German, 7 % were Dutch, and other participants came from the USA, China, South Africa, Israel, and other European countries; overall 13 different countries were represented.

A restricted number of travel grants was available for students. They were granted after a selection process based on the submission of a resume. Six students benefited from these grants.

Wishing to be open to all researchers, registration to EWRL was free. The organization was funded by the SequeL team project of INRIA, the "Collège Doctoral Européen Lille Nord-Pas de Calais," who funded the travel grants, the INRIA Research Center "Lille-Nord Europe," and the computer science laboratory (LIFL) of the University of Lille. EWRL benefited from their support, and we would like to take the opportunity to acknowledge this help, and thank them for it.

EWRL 2008 would not have been a success without the help of many people. First of all, we would like to thank all the contributors to the workshop. Among them, we would like to offer special thanks to the three invited speakers who readily accepted our invitation. The reviewers and the Program Committee did a great job, which resulted in the program of the workshop and the papers selected for this volume. The participants of the workshop made it a great, lively scientific event. Many people worked behind the scenes: we would like to thank the "École Centrale de Lille" (ECL) that hosted EWRL, its Secrétaire Général, Mr. Parisis, who facilitated many things on the practical side, and the staff of the ECL. The secretary of the workshop, Sandrine Catillon, from INRIA did a great deal of work with skill and diligence; the management of the travel grants has been magnificantly handled by Hélène Fourmentraux from the LIFL. We would also like to take the opportunity to thank EasyChair, the free conference management system, which managed the whole process of submission and reviewing, up to the creation of the draft of the proceedings, according to Springer requirements; Easy Chair definitely saved us a lot of time.

Finally, we would also like to thank Springer for their early support in this endeavor; Springer promptly accepted to publish these proceedings in their re-known *Lecture Notes in Artificial Intelligence* series while the organization of EWRL 2008 was still in a very early stage, and its success was still unsure.

September 2008

Sertan Girgin
Manuel Loth
Rémi Munos
Philippe Preux
Daniil Ryabko

Organization

Program Chairs

Sertan Girgin
Manuel Loth
Rémi Munos
Philippe Preux
Daniil Ryabko

Program Committee

Peter Auer
Sébastien Bubeck
Rémi Coulom
Peter Dayan
Kurt Driessens
Alain Dutech
Yaakov Engel
Damien Ernst
Frédérick Garcia
Mohammad Ghavamzadeh
Daniel Kudenko
Michail G. Lagoudakis
Pier Luca Lanzi
Jérémie Mary
Francisco Melo
Ann Nowé
Jan Peters
Olivier Pietquin
Martin Riedmiller
Régis Sabbadin
Bruno Scherrer
Olivier Sigaud
Csaba Szepesvari
Chris Watkins
Shimon Whiteson
Marco Wiering
Jeremy Wyatt
Martin Zinkevich
Martijn van Otterlo

Table of Contents

Lazy Planning under Uncertainty by Optimizing Decisions on an Ensemble of Incomplete Disturbance Trees

Boris Defourny, Damien Ernst, and Louis Wehenkel

University of Liège,
Department of Electrical Engineering and Computer Science,
Grande Traverse, 10, Sart-Tilman, B-4000 Liège, Belgium
{Boris.Defourny,dernst,L.Wehenkel}@ulg.ac.be

Abstract. This paper addresses the problem of solving discrete-time optimal sequential decision making problems having a disturbance space W composed of a finite number of elements. In this context, the problem of finding from an initial state x_0 an optimal decision strategy can be stated as an optimization problem which aims at finding an optimal combination of decisions attached to the nodes of a *disturbance tree* modeling all possible sequences of disturbances $w_0, w_1, \ldots, w_{T-1} \in W^T$ over the optimization horizon T. A significant drawback of this approach is that the resulting optimization problem has a search space which is the Cartesian product of $O(|W|^{T-1})$ decision spaces U, which makes the approach computationally impractical as soon as the optimization horizon grows, even if W has just a handful of elements. To circumvent this difficulty, we propose to exploit an ensemble of randomly generated *incomplete* disturbance trees of controlled complexity, to solve their induced optimization problems in parallel, and to combine their predictions at time $t = 0$ to obtain a (near-)optimal first-stage decision. Because this approach postpones the determination of the decisions for subsequent stages until additional information about the realization of the uncertain process becomes available, we call it *lazy*. Simulations carried out on a robot corridor navigation problem show that even for small incomplete trees, this approach can lead to near-optimal decisions.

Keywords: Stochastic dynamic programming, Ensemble methods.

1 Introduction

The discrete-time optimal control paradigm can be used to formalize a broad class of problems arising in a variety of fields such as finance, automatic control, robotics, or operations research. In this paradigm, at each time step t, the decision maker measures the state $x_t \in X$ of the environment and takes a decision $u_t \in U$ according to his decision rule. As a result of his decision, the environment transits to a new state x_{t+1} and the decision maker observes a scalar reward signal r_t which reflects in some way the impact of his decision on his performance criterion. The environment behavior as perceived by the decision maker can be in general nonlinear and stochastic.

In this paper, we assume that the stochastic nature of the environment is modeled by an *exogenous* disturbance process ($w_t \in W$) which acts as an additional input on the state transitions. We also assume that this process is memoryless and that the optimality criterion J of the decision maker is the expected value of the sum of rewards r_t over a finite number T of stages. Within this context, we focus on the computation of decisions that maximize J when a model of the environment is provided to the decision maker in terms of the initial condition x_0, a full specification of the stochastic process w_t, and the functions $f_t(\cdot,\cdot,\cdot)$ and $r_t(\cdot,\cdot,\cdot)$ allowing to compute x_{t+1} and r_t from x_t, u_t and w_t.

Different strategies can be thought of for computing decisions in such a context. One of them would be to compute a fixed sequence of decisions u_0, u_1, $\ldots$, u_{T-1} from the available information. Solving the problem in this manner is equivalent to searching in the space $U^T = \times_{t=0}^{T-1} U$ an element that maximizes the optimality criterion. This approach, also known as the *open-loop* approach, is very convenient for solving problems for which the dynamics of the environment is linear and deterministic, and the optimality criterion J convex. Indeed, under these conditions, efficient algorithms for searching for the best element in U^T exist, and it can be shown that among the set of all plausible decision rules, the one deduced by such an approach is optimal within this restricted context [1,2].

Putting aside the difficulty of solving this optimization problem for more general classes of problems, the open-loop approach is intrinsically less attractive for stochastic problems since the sequence of decisions so obtained is generally quite suboptimal with respect to the use of optimal closed-loop decision rules, i.e. decision rules which determine the current action from the current time and current information available about the state of the environment. While the suboptimality of the open-loop approach could to some extent be decreased if at time $t > 0$, the last remaining $T - t$ decisions were reoptimized by taking into account that the system is in state x_t at time t, something it was impossible to predict at times $0, 1, \ldots, t-1$ due to the stochasticity of the system, the inherent suboptimality of such a time receding open-loop approach remains because the stochastic nature of the problem is not explicitly taken into account when computing the sequence of decisions.

Actually, some approaches exist to extend in an optimal way the open-loop philosophy to stochastic problems [3]. For easing subsequent discussions, we will suppose that the disturbance space W of the stochastic optimal control problem is finite. The main philosophy behind these approaches is the following. First, they generate all the $|W|^T$ possible T-stage disturbance scenarios w_0, w_1, $\ldots$, w_{T-1}. Secondly, they associate to each one of these scenarios and each possible sequence of decisions a return. Then they determine by solving in one single optimization problem $|W|^T$ sequences of decisions of length T, one for each scenario, such that the expected return over all the scenarios is maximized. By imposing the constraint that the first t decisions of two such sequences corresponding to two scenarios having the same $t-1$ first elements w are the same, these techniques can be used to determine optimal decision rules which associate a decision to each time step t and any partial sequence of disturbances $w_0, \ldots, w_{t-1}$. An immediate variant of this approach which represents implicitly the equality constraints over the decisions (called non-anticipativity constraints) is obtained by reformulating the optimization problem on a disturbance tree.

One main drawback of these disturbance tree approaches for solving stochastic optimal control problems is related to the size of the optimization problem they require to solve. Indeed, the search space for these optimization problems is the Cartesian product of n decision spaces U where n is $O(|W|^{T-1})$. This exponential growth makes the approach rapidly computationally impractical as soon as T grows, even if W has only a few elements.

In order to circumvent this difficulty, we propose in this paper an alternative approach exploiting an ensemble of randomly generated *incomplete* disturbance trees of controlled complexity. In this approach, the incomplete disturbance trees are built by developing each node only partially, that is by not necessarily associating to a non-terminal node $|W|$ children nodes. More specifically, our strategy for selecting the disturbances for developing a node is nondeterministic and tends to develop less the nodes as the tree depth increases. This strategy depends on parameters that control the expected number of nodes of a tree, and, consequently, which influence the size of the search space for the optimization problem since it is equal to the Cartesian product of n decision spaces U, where n is the number of nonterminal nodes of the tree. The optimization problems induced by these trees can be solved in parallel. Their predictions at time $t = 0$ are combined in order to compute a (near-)optimal first-stage decision for the problem at hand. We call this approach *lazy*, in order to stress the fact that it postpones the determination of the decisions for subsequent stages until additional information about the realization of the uncertain process becomes available[1]. We provide simulation results carried out on a robot navigation problem which suggest that the proposed framework can strongly improve the computational performances of the original disturbance tree approach for planning under uncertainty while being only slightly suboptimal.

The rest of the paper is structured as follows. Section 2 discusses our "ensemble of incomplete disturbance trees" approach with respect to different works in planning under uncertainty and machine learning. In Section 3, we specify the type of planning under uncertainty problem considered in this paper and show how the problem of finding an optimal decision strategy can be reformulated as an optimization problem on a disturbance tree. This section also describes an example of application of this technique to a robot navigation problem when solving the optimization problem by using a Cross-Entropy based algorithm. Section 4 describes our ensemble of incomplete disturbance trees approach for deriving the first-stage decision and evaluates its performances on the robot navigation benchmark problem. Finally, Section 5 concludes.

[1] The term "lazy" is used in compiler theory and machine learning in order to qualify algorithms which delay computations until enough information is available to decide that they indeed need to be carried out [4,5].

2 Related Work

The paradigm of optimizing decisions on a disturbance tree has already been studied by several authors. Most of their works have been carried out by assuming that the system dynamics is linear and by supposing as disturbance spaces compact subsets of $\mathbb{R}^n$. One central theoretical result in this field is that as the discretization of the disturbance space becomes finer, the suboptimality of the approach decreases to zero [6,7]. Approaches proposed for building the disturbance trees have been diverse but the central motivation behind them is always the same: having a small disturbance tree which leads to a near-optimal decision rule. As different strategies used for building these trees, one can mention those which interlace the building of the tree with the optimization process [8] or those which optimize decision variables once the tree is built. For these latter ones, one distinguishes methods based on Monte Carlo sampling [9], on the preservation of the statistical properties [10], and on the minimization of probability metrics between the target and the approximate distribution [11,12].

When the disturbance space is composed of a single element (deterministic environment), these disturbance tree approaches degenerate into the computation of an open-loop sequence of decisions. The computation of such sequences of decisions is at the heart of the vastly successful Model Predictive Control techniques which combine their computation with some time receding horizon strategies [1,2,13]. These techniques have also been extended to stochastic environments but rather by using some min-max approaches aimed at finding a solution which is optimal with respect to the worst-case "disturbance sequence" [14], rather than by trying to maximize a compound return function J.

The approach proposed in this paper for alleviating the computational burdens related to the disturbance tree paradigm is the first one which proposes to build an ensemble of models and to aggregate their solution, with the exception perhaps of the work of Nesterov and Vial, who develop in [15] similar ideas for highly structured environments. However, the idea of aggregating the individual outputs of an ensemble of models has already been vastly exploited in machine learning, and especially in supervised learning (classification and regression). As way of example, one can mention the boosting method [16] which builds models sequentially and refines the output regions where the errors are important or the bagging method which builds the models in parallel from some randomized sets of data [17]. These ideas of aggregating the predictions of an ensemble of models have also already been used for planning under uncertainty but in the context where closed-loop decision rules are computed [18,19].

Finally, we mention the work of Kearns et al. [20] who propose to solve stochastic planning problems by developing a tree where each branch corresponds to a decision-disturbance pair. They apply the dynamic programming principle on the tree to compute decisions and show under some particular conditions that sparse sampling of the disturbances suffices to compute near-optimal decisions. As in our approach, the complexity of their tree does not depend on the number of states. However, and contrary to the disturbance tree approach, their tree size grows (rapidly) with the cardinality of the set of possible actions U.

3 Planning over a Disturbance Tree

3.1 Formalization

Our first task will be to formulate a sequential decision problem over a time horizon T, and to explain how a decision making strategy can be evaluated on a complete disturbance tree of depth T.

We consider a system that evolves according to a state transition function $x_{t+1} = f_t(x_t, u_t, w_t)$ starting from a fixed initial state x_0. Its trajectories are controlled by the decisions $u_t \in U$ and perturbed by disturbances $w_t \in W$, which are drawn independently from a finite time-invariant probability distribution $\mathbb{P}_w$[2]. A reward process $r_0, r_1, \ldots, r_T$ is defined by $r_t = r_t(x_t, u_t, w_t)$ for $0 \leq t < T$ and $r_T = r_T(x_T)$. The goal is to find a strategy μ maximizing the expectation of a discounted sum of the rewards, with $0 < \gamma \leq 1$ the discount factor[3]:

$$J^*(x_0) = \max_{\mu} \mathbb{E} \left\{ \sum_{t=0}^{T-1} \gamma^t r_t(x_t, u_t, w_t) + \gamma^T r_T(x_T) \right\} . \qquad (1)$$

In the disturbance tree framework, the strategy μ for selecting a decision u_t at time $0 \leq t < T$ consists in deterministic mappings μ_t from current histories $h_t = [w_0, w_1, \ldots, w_{t-1}]$ of the disturbance process to decisions u_t at time t. The mapping at time 0 degenerates into a fixed decision u_0, the history at time 0 being empty. This class of strategies is in principle more general than the class of time-dependent strategies mapping the state x_t to a decision u_t, since the state x_t can always be recovered by the procedure described in Table 1. However, when the disturbance process is memoryless, these two classes of strategies are equivalent in terms of optimality.

Table 1. How to recover states from disturbance histories

Input: An initial state x_0, a history of the disturbance process $h_t = [w_0, w_1, \ldots, w_{t-1}]$, a strategy $\mu_0, \mu_1, \ldots, \mu_{t-1}$ for computing decisions up to time $t-1$.
Output: The state x_t.

1. Initialization: Set x to x_0 and τ to 0.
2. While $\tau < t$, set u to $\mu_\tau(w_0, w_1, \ldots, w_{\tau-1})$, set x to $f_\tau(x, u, w_\tau)$, and increment τ.
3. Return x.

The complete disturbance tree represents all the possible outcomes of the process $w_0, w_1, \ldots, w_{T-1}$. Its construction is given in Table 2. To each node n of depth $0 < t \leq T$ in the tree corresponds a history $h_n = [w_0, \ldots, w_{t-1}]_n$ of the process, through the unique path from the root to the node n. The disturbance

[2] Independence of the w_t is imposed only to simplify the presentation and facilitate the parallel with the dynamic programming framework.

[3] More general objective functions could also be considered in this framework.

Table 2. How to build a complete disturbance tree

Input: The horizon T, the disturbance space $W = \{W_1, \ldots, W_m\}$, the probabilities $P_j = \mathbb{P}(w_t = W_j)$.
Output: A complete disturbance tree over the finite horizon T.

```
1. Initialization : Create a root node of depth 0.
   Associate the probability 1 to the root. Set $t = 0$.
2. While $t < T$:
   For each node $n$ of depth $t$, create $m$ successor nodes (of depth $t+1$) with associated
   values $W_1, \ldots, W_m$ and probabilities $P_1, \ldots, P_m$.
   Increment $t$.
3. Return the tree structure and the values $w_n$ and probabilities $p_n$ associated to its
   nodes $n$. (Now $w_n$ denotes a value of $w_t$, where $t$ is the depth of node $n$.)
```

Table 3. Evaluation of the expected value of a strategy on a disturbance tree

Input: A disturbance tree, an initial state x_0, a strategy μ represented by decisions u_n associated to the nodes n of depth $0 \leq t < T$.
Output: The expected value of the decision making strategy.

```
1. (Computation of the rewards $r_n$ associated to the nodes of the tree.)
   Associate the initial state $x_0$ to the root node. Set $t$ to 1.
   While $t \leq T$ :
   For each node $n$ of depth $t$: Identify the parent node $n'$ and associate
   $x_n = f_{t-1}(x_{n'}, u_{n'}, w_n)$ and $r_n = r_{t-1}(x_{n'}, u_{n'}, w_n)$ to node $n$.
   Increment $t$.
2. (Computation of the expected discounted sum of rewards by backpropagation.)
   Associate $J_n = r_T(x_n)$ to each node $n$ of depth $T$. Set $t$ to $T-1$.
   While $t \geq 0$:
   For each node $n$ of depth $t$: Identify the set of successor nodes $S(n)$, and associate
   $J_n = \sum_{n' \in S(n)} p_{n'} \cdot (r_{n'} + \gamma J_{n'})$ to node $n$.
   Decrement $t$.
3. Return the value $J_n$ associated to the root node.
   Clearly, this value is equal to $\mathbb{E}\{\sum_{t=0}^{T-1} \gamma^t r_t(x_t, u_t, w_t) + \gamma^T r_T(x_T)\}$
   where $u_t = \mu_t(w_0, \ldots, w_{t-1})$.
```

$(w_{t-1})_n$ is directly associated to node n, while $[w_0, \ldots, w_{t-2}]_n$ can be collected from the disturbances associated to the nodes in the path. The root, at $t = 0$, has an empty history. The strategy μ can thus be represented on the tree by associating to each node n of depth $0 \leq t < T$ the value $u_n = \mu(h_n)$.

Alternatively, searching for an optimal strategy becomes equivalent to optimizing the values u_n. To this end, states x_n, rewards r_n and partial sums J_n are associated to nodes n. This helps to compute the expected value of an arbitrary strategy μ represented by particular values for u_n. Table 3 describes the full process. The optimization itself is done directly over the node variables u_n. The algorithm of Table 3 will serve as an oracle for scoring the strategy.

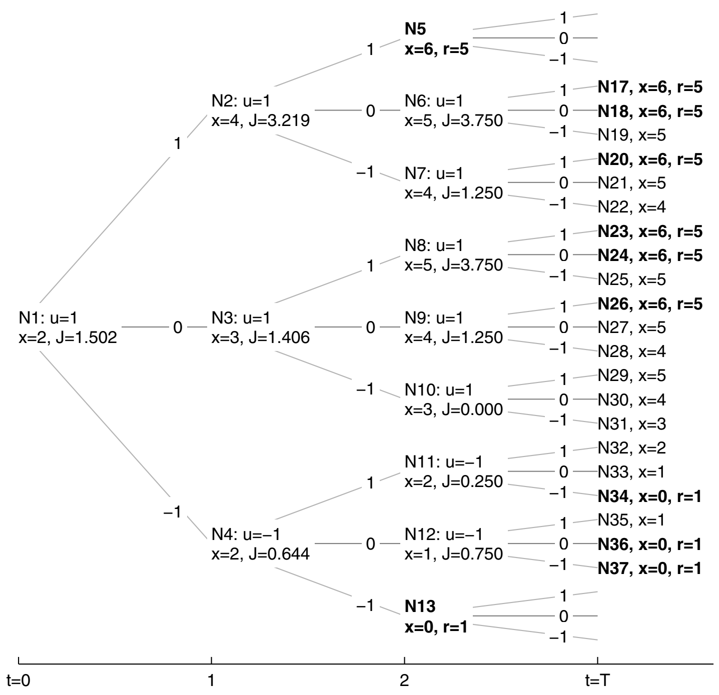


Fig. 1. A toy corridor problem on $T = 3$ with terminal states $\{0, 6\}$, starting from $x_0 = 2$, solved on a complete disturbance tree. The node disturbances w_n are written on the branches and the corresponding node probabilities ($p_n = 0.25$ for $w_n = \pm 1$, $p_n = 0.5$ for $w_n = 0$) are omitted. The values u_n have been optimized, and this fully specifies an optimal strategy μ. For instance, $u = -1$ at node N11 means that $\mu_2(w_0 = -1, w_1 = +1) = -1$. A terminal state is reached at node N5, among other cases. This makes the subtree beyond N5 useless, but only once the decisions are optimized. The expected value of the strategy, $J = 1.502$, is read from the root node N1. Reported values for x_n, r_n, J_n are those obtained with the last invocation of the algorithm of Table 3, which served to score decisions u_n's generated by the Cross-Entropy method. For instance, the value $J = 0.75$ at node N12 comes from $p_{\mathrm{N36}} r_{\mathrm{N36}} + p_{\mathrm{N37}} r_{\mathrm{N37}}$, in accordance with the step 2 of Table 3.

3.2 Illustration

The following corridor navigation problem illustrates the search for an optimal strategy on a complete disturbance tree.

Let $X = \{1, 2, \ldots, S-1\} \cup X_{\mathrm{term}}$, where $X_{\mathrm{term}} = \{0, S\}$ is a set of terminal states. Let $W = \{-1, 0, 1\}$, with probabilities $\mathbb{P}_w = \{0.25, 0.50, 0.25\}$.

If $x_t \notin X_{\mathrm{term}}$, then $U = \{-1, +1\}$ and

$$\begin{cases} x_{t+1} = x_t + u_t + w_t, & r_t = 0 \quad \text{if } 0 < x_t + u_t + w_t < S \\ x_{t+1} = 0, & r_t = 1 \quad \text{if } x_t + u_t + w_t \leq 0 \\ x_{t+1} = S, & r_t = 5 \quad \text{if } x_t + u_t + w_t \geq S. \end{cases}$$

If $x_t \in X_{\mathrm{term}}$, then $x_{t+1} = x_t$, $r_t = 0$, and U is irrelevant.

The terminal rewards of equation (1) are set to $r_T(\cdot) = 0$.

In this section, $T = 3$, $S = 6$, $\gamma = 0.9$, and $x_0 = 2$.

The complete disturbance tree is represented on Fig. 1, along with the optimized decisions u_n and the corresponding states x_n, rewards r_n, and partial sums J_n. The Cross-Entropy method was used to find a global optimal solution for u_n. The reader may refer to [21] for more details on the Cross-Entropy method, which was well adapted for the particular problem at hand. Simply put, the Cross-Entropy method samples candidate solutions, using importance sampling oriented towards candidate solutions with the highest scores. If the method succeeds, then the output of Table 3 corresponds to (1). Otherwise, the suboptimal solution that has been found may still be acceptable.

The decision making strategy reported on the figure is indeed optimal. The value $J = 1.502$ corresponds to the value returned by the well-known value iteration algorithm from dynamic programming, stopped after $T = 3$ iterations.

One can check on Fig. 1 that the mapping from states x_t to decisions u_t (which is the kind of mapping considered in the value iteration algorithm) corresponding to the history-to-decision mapping found on the disturbance tree is time-variant[4].

4 Lazy Approach Using an Ensemble of Incomplete Disturbance Trees to Derive a First-Stage Decision

4.1 Principle

Our second task will be to explain how we can advantageously work on an ensemble of incomplete disturbance trees to determine an optimal first-stage decision.

The representation of a strategy μ by decisions u_n defined on the nodes of the complete disturbance tree shows that an incomplete disturbance tree will only permit an incomplete description of a strategy. However, we restrict our attention to the first-stage decision u_0, and assume that the optimization of incomplete strategies will give some valuable information on u_0. We thus propose to aggregate several first-stage decisions obtained from several incomplete strategies optimized on several incomplete disturbance trees. The full process, based on a majority vote for the decision u_0 and described in Table 4, can be restarted at each time step, with the current state as the initial condition.

An incomplete disturbance tree can be built using the randomized algorithm given in Table 5. Starting from the root, the algorithm creates for each node a set of successor nodes by sampling disturbances. Distinct samples form the successor nodes, while the sample multiplicities serve to define node probabilities. The growth of the tree is controlled by choosing a random number m of samples to draw.

The probability distribution of m is an input for the algorithm. For building deeper trees while preserving the branching structure allowed on shorter time

[4] For instance the nodes N1 ($t = 0$) and N4 ($t = 1$) share the state $x = 2$ but differ in the decision.

Table 4. Computing a decision at time $t = 0$ with an ensemble of M incomplete trees

Input: The initial state x_0, the optimization horizon T, the desired number M of trees, a specification of the disturbance process $w_{t=0}^{T-1}$.
Output: A decision u_0 to be applied at time 0.

1. Build in parallel M incomplete disturbance trees of depth T.
2. Optimize in parallel a decision making strategy on each tree.
3. Return the decision obtained by a majority vote over the M first-stage decisions u_0 gathered from the roots of the M trees.

Table 5. How to build an incomplete disturbance tree for a discrete, time-invariant, and memoryless disturbance process

Input: The disturbance space W, the probabilities $\mathbb{P}_w$, the tree depth T.
Parameters: Probability distributions $\mathbb{Q}_A$, $\mathbb{Q}_B$, both of support in $\{1, 2, \ldots, q\}$
where q is the maximal allowed number of samples.
Output: An incomplete disturbance tree over the horizon T.

1. Initialization : Create a root node of depth 0.
   Associate the probability 1 to the root. Set $t = 0$.
2. While $t < T$:
   Set α to $1/(1+t)$. Set $\mathbb{Q}_t$ to $\alpha\mathbb{Q}_A + (1-\alpha)\mathbb{Q}_B$.
   For each node n of depth t:
   Draw a random number m according to $\mathbb{Q}_t$. Draw m samples in W according to $\mathbb{P}_w$.
   Obtain $m' \leq m$ distinct samples $[w^{(1)}, \ldots, w^{(m')}]$ of multiplicity $[k^{(1)}, \ldots, k^{(m')}]$.
   Create m' successor nodes (of depth $t+1$) with associated values $w^{(1)}, \ldots, w^{(m')}$ and probabilities $k^{(1)}/m, \ldots, k^{(m')}/m$.
   Increment t.
3. Output the incidence structure of the tree, and the values w_n and probabilities p_n associated to each one of its nodes n.

horizons, it might be advantageous to let the distribution of m evolve with the depth of the tree. A simple way to do that consists in mixing 2 probability distributions $\mathbb{Q}_A$ and $\mathbb{Q}_B$ with relative weights depending on the depth. More specifically, the probability that $m = j$ for a node of depth t is set to

$$\mathbb{Q}_t(m = j) = \alpha\mathbb{Q}_A(m = j) + (1-\alpha)\mathbb{Q}_B(m = j) \tag{2}$$

with $\alpha \triangleq 1/(1+t)$ moving progressively from 1 to 0 as the depth t increases. Trees are likely to feature sequences of disturbances with few branchings beyond a certain depth when $\mathbb{Q}_B$ is concentrated on small values of m such as 1 or 2. Of course, the algorithm can be run with fixed branching probabilities by setting $\mathbb{Q}_A \equiv \mathbb{Q}_B$. However, if $\mathbb{Q}_B$ has its mass concentrated on 1, the expected number of nodes is asymptotically linear with the depth T of the tree. This strongly contrasts with the exponential growth of the size of the complete tree with T.

It is easy to estimate by Monte Carlo simulation the expected size of an incomplete tree for given depth T, number of disturbances $|W|$ and probabilities $\mathbb{Q}_A$,

$\mathbb{Q}_B$. Therefore, it is possible to choose $\mathbb{Q}_A$ and $\mathbb{Q}_B$ so that the expected size of a tree is in line with the size of the optimization problem one is able to deal with.

4.2 Illustration

The corridor navigation problem of Section 3 will illustrate the clear benefit in terms of computational complexity of working on an ensemble of incomplete disturbance trees instead of a unique complete tree. A set of simulations is run with $S = 10$, $T = 6$, $\gamma = 0.7$, on different initial conditions x_0.[5]

The problem is solved for $x_0 = 3$ on a complete tree (composed of 1093 nodes) in a reasonable time. This permits to report relative time savings for the solution on incomplete trees. A majority vote over $M = 100$ incomplete trees gives the first decision u_0. Subsequent decisions are ignored, assuming that simpler problems on shorter time horizons will be solved to obtain u_1, u_2, ... as soon as x_1, x_2, ... are known.

Table 6 gives the 4 representative choices for the distributions $\mathbb{Q}_A$ and $\mathbb{Q}_B$ that have been considered as parameters of the tree generation algorithm, along with their macroscopic effect on the size of a tree of depth $T = 6$.

Table 7 shows that with the first choice (Test I), the trees reduce to T-length sequences of disturbances. Wrong decisions for $x_0 = 4$ and $x_0 = 5$ indicate that this structure is too weak. Test IV leads to moderately dense trees and right decisions. Beside these two extremes, Test II uses fixed probabilities for the number of disturbances to sample and Test III use decaying probabilities. The setting for Test III dominates the one for Test II, because it produces smaller trees while these trees prove more reliable on the initial condition $x_0 = 4$: they advocate the optimal decision more often.

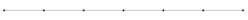
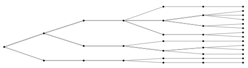
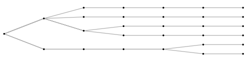
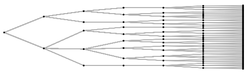
Table 6. Representative settings for the probability distributions of the number of samples per node, and their effect on the size of the incomplete trees

	Q_A [†]	Q_B	Short description	Number of nodes [‡]	
				Mean	Std dev.
I	[1 0 0]	[1 0 0]	Always draw 1 sample.	7	0
II	[1 1 1]/3	[1 1 1]/3	1,2,3 samples with prob. 1/3.	39.31	22.31
III	[0 0 1]	[1 0 0]	Deeper, draw 1 instead of 3.	29.87	13.10
IV	[0 0 1]	[0 0 1]	Always draw 3 samples.	136.25	53.28

[†] $Q_A = [1\ 0\ 0]$ means that the probability of drawing 1, 2, 3 samples is respectively 1, 0, 0 under the distribution $\mathbb{Q}_A$ that parameterizes the algorithm of Table 5. The number of children nodes may be less, depending on the number of distinct samples.
[‡] Estimated by building 400 trees of depth $T = 6$. A complete tree has 1093 nodes.

[5] We used a larger optimization horizon T than in Section 3 to better highlight the performances of our approach. The value of γ differs also from the one previously chosen (now 0.7 rather than 0.9) to have an optimal first-stage decision that varies with the initial state, so as to make the example more interesting.

Table 7. Decisions obtained with incomplete trees on the corridor problem with terminal states $\{0, 10\}$, $T = 6$, and $\gamma = 0.7$. The 4 settings of Table 6 are tested. Basically, decisions are correct when the trees are not too small.

	Decision $u_0^\dagger$				Time[‡]	Typical tree
	$x_0 = 2$	$x_0 = 3$	$x_0 = 4$	$x_0 = 5$		
Optimal:	*-1*	*-1*	*+1*	*+1*		
Test I	-1 (93%)	-1 (78%)	-1 (66%)	-1 (51%)	0.04%	
Test II	-1 (96%)	-1 (82%)	+1 (54%)	+1 (81%)	0.42%	
Test III	-1 (95%)	-1 (80%)	+1 (63%)	+1 (88%)	0.15%	
Test IV	-1 (98%)	-1 (91%)	+1 (71%)	+1 (100%)	2.41%	

[†] Initial decision obtained by a majority vote on $M = 100$ trees, with the share in parenthesis. Separate tests are conducted for the 4 mentioned initial states x_0.
[‡] Mean computation time for solving 1 incomplete tree (case $x_0 = 3$), in percentage of the time required to solve the complete disturbance tree for the same problem. Time savings are huge.

Beyond the huge time savings, the tests have also revealed an unexpected advantage of using the Cross-Entropy method on incomplete trees. Suboptimal solutions indeed arise more frequently on the complete tree than on the smaller incomplete ones, by a simple scale effect. A wrong decision can thus be inferred from a suboptimal solution on the complete tree, while the majority vote from solutions on incomplete trees yields a correct decision in a more reliable way.

5 Conclusions

This paper has proposed an approach for alleviating the computational burdens of the original disturbance tree paradigm for planning under uncertainty. The approach builds an ensemble of small trees, solves in parallel the small size optimization problems which correspond to these trees and aggregates their first-stage decisions. Applying this algorithm in a time receding fashion results in a *lazy* decision making strategy which computes at each time step the decision to apply given all the available information, thereby focusing only on the particular subproblem to solve rather than trying to pre-compile once and for all a decision strategy that would be applicable to all kinds of realizations of the environment under control. This approach has been evaluated on a robot navigation problem and the results obtained are encouraging. Indeed, with an ensemble of small trees (especially with respect to the fully developed one), it has been possible to obtain in a reliable way accurate predictions of the optimal first-stage decision.

While the strategy adopted in this paper for building the incomplete trees is giving good results, we do not exclude that it could still be significantly improved by for example relying on other sampling procedures than a pure Monte Carlo one for developing a disturbance tree node. It would also be interesting to study to which extent some specific optimization tools could be customized to the structure of the optimization problem defined by a disturbance tree for leveraging their performances. Besides the study of the algorithmic improvements that could be brought to our multi-tree framework, it would also be interesting to establish its theoretical properties. For example, it would be informative to know (even under some highly restrictive assumptions on the environment structure) some upper bounds (in probability) on the suboptimality of the decision rules with respect to the size of the ensemble, the variance of the computed decisions or the computational complexity of these multi-tree based algorithms.

Also, while we have in this paper worked out the approach in the context of a memoryless disturbance process, in which case the state of the system under control is a sufficient statistic to take decisions, our approach extends in a straightforward way to general disturbance processes, or in other words to partially observable environments. Further work should thus be carried out to study in more depth the pros and cons of this approach with respect to the literature on partially observable Markov decision processes [22,23].

Another important direction of research concerns the extension of this approach to continuous disturbance processes. Some preliminary work [24,25] along these lines shows that in this context it is important to properly choose the way in which the continuous disturbance process is discretized in order to generate the disturbance trees.

Overall, the approach presented in this paper adds to the arsenal of methods for planning under uncertainty. However, it is not yet clear how it would compete for some specific classes of problems with algorithms exploiting other paradigms, such as the dynamic programming paradigm [26] or the direct closed-loop policy search one [27]. All these paradigms have pros and cons, and establishing which one, or which combination of paradigms, to exploit for a specific problem certainly deserves further research.

Acknowledgments. Damien Ernst acknowledges the financial support of the Belgian National Fund of Scientific Research (FNRS) of which he is a Research Associate. This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors.

References

1. Maciejowski, J.: Predictive Control with Constraints. Prentice Hall, Englewood Cliffs (2001)
2. Morari, M., Lee, J.: Model predictive control: past, present and future. Computers and Chemical Engineering 23, 667–682 (1999)

3. Birge, J., Louveaux, F.: Introduction to Stochastic Programming. Springer, New York (1997)
4. Launchbury, J.: A natural semantics for lazy evaluation. In: POPL 1993: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 144–154. ACM, New York (1993)
5. Friedman, J., Kohavi, R., Yun, Y.: Lazy decision trees. In: Proc. of 13th National Conference on Artificial Intelligence, AAAI 1996. Part 1(of 2), pp. 717–724 (1996)
6. Heitsch, H., Römisch, W., Strugarek, C.: Stability of multistage stochastic programs. SIAM Journal on Optimization 17(2), 511–525 (2006)
7. Römisch, W.: Stability of stochastic programming problems. In: Ruszczyński, A., Shapiro, A. (eds.) Stochastic Programming. Handbooks in Operations Research and Management Science, vol. 10, pp. 483–554. Elsevier, Amsterdam (2003)
8. Dempster, M.: Sequential importance sampling algorithms for dynamic stochastic programming. Annals of Operations Research 84, 153–184 (1998)
9. Shapiro, A.: Monte Carlo sampling methods. In: Ruszczyński, A., Shapiro, A. (eds.) Stochastic Programming. Handbooks in Operations Research and Management Science, vol. 10, pp. 353–425. Elsevier, Amsterdam (2003)
10. Høyland, K., Wallace, S.: Generating scenario trees for multistage decision problems. Management Science 47(2), 295–307 (2001)
11. Hochreiter, R., Pflug, G.: Financial scenario generation for stochastic multi-stage decision processes as facility location problems. Annals of Operations Research 152, 257–272 (2007)
12. Rachev, S., Römisch, W.: Quantitative stability in stochastic programming: The method of probability metrics. Mathematics of Operations Research 27(4), 792–818 (2002)
13. Ernst, D., Glavic, M., Capitanescu, F., Wehenkel, L.: Reinforcement learning versus model predictive control: a comparison on a power system problem. IEEE Transactions on Systems, Man and Cybernetics - Part B (to appear, 2008)
14. Kothare, M., Balakrishnan, V., Morari, M.: Robust constrained model predictive control using matrix inequalities. Automatica 32, 1361–1379 (1996)
15. Nesterov, Y., Vial, J.P.: Confidence level solutions for stochastic programming. Automatica 44(6), 1559–1568 (2008)
16. Schapire, R.: The strength of weak learnability. Machine Learning 5(2), 197–227 (1990)
17. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
18. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556 (2005)
19. Sutton, R.: Generalization in reinforcement learning: successful examples using sparse coarse coding. Advances in Neural Information Processing Systems 8, 1038–1044 (1996)
20. Kearns, M., Mansour, Y., Ng, A.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Machine Learning 49(2-3), 193–208 (2002)
21. Rubinstein, R., Kroese, D.: The Cross-Entropy Method. A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning. In: Information Science and Statistics. Springer, Heidelberg (2004)
22. Cassandra, A., Kaelbling, L., Littman, M.: Acting optimally in partially observable stochastic domains. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994), Seattle, Washington, USA, vol. 2, pp. 1023–1028. AAAI Press/MIT Press, Menlo Park (1994)

23. Ng, A., Jordan, M.: PEGASUS: a policy search method for large MDPs and POMDPs. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 406–415 (1999)
24. Defourny, B.: Approximate solution to multistage stochastic programs with ensembles of randomized scenario trees. Master's thesis, University of Liège, Department of Electrical Engineering and Computer Science (2007)
25. Defourny, B., Wehenkel, L.: Averaging decisions from an ensemble of scenario trees: a validation on newsvendor problems (submitted, 2008)
26. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
27. Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Advances in Neural Information Processing Systems 12, 1057–1063 (2000)

Exploiting Additive Structure in Factored MDPs for Reinforcement Learning

Thomas Degris, Olivier Sigaud, and Pierre-Henri Wuillemin

Université Pierre et Marie Curie - Paris6
4 place Jussieu, F-75005 Paris, France
thomas.degris@laposte.net, Olivier.Sigaud@lip6.fr,
Pierre-Henri.Wuillemin@lip6.fr

Abstract. SDYNA is a framework able to address large, discrete and stochastic reinforcement learning problems. It incrementally learns a FMDP representing the problem to solve while using FMDP planning techniques to build an efficient policy. SPITI, an instantiation of SDYNA, uses a planning method based on dynamic programming which cannot exploit the additive structure of a FMDP. In this paper, we present two new instantiations of SDYNA, namely ULP and UNATLP, using a linear programming based planning method that can exploit the additive structure of a FMDP and address problems out of reach of SPITI.

1 Introduction

Markov Decision Processes (MDPs) are a fundamental framework to model planning under uncertainty problems as well as Reinforcement Learning (RL) problems. Standard exact solution methods for both problems are known to work well but are inappropriate for large problems because they require explicit state space enumerations. Among different approximation techniques, factored MDPs (FMDPs), first proposed by [1], assume the decomposition of the state space with random variables. FMDPs utilize dependencies between variables, defined using Dynamic Bayesian Networks (DBNs) [2], to compactly represent the transition and reward functions of structured MDPs.

When the structure of the transition and reward functions are fully known, solution methods may be used to compute optimal (or near optimal) value functions and policies of the FMDP. These solution methods are based on two different classical techniques, namely Dynamic Programming (DP) and Linear Programming (LP). First, algorithms such as Structured Policy Iteration (SPI), Structured Value Iteration (SVI) and Stochastic Planning Using Decision Diagrams (SPUDD) are based on DP [3,4] and mainly exploit context specific independence by using structured representations (i.e. decision trees or decision diagrams) to manipulate the functions of the FMDPs. Second, [5] proposes different algorithms based on LP that can exploit additional regularities such as the additive decomposition of the reward function and an additive approximation of the value function. We name such regularities as additive structure of the problem.

When the structure of the transition and reward functions are unknown, [6] has proposed Structured DYNA (SDYNA). SDYNA is a general framework combining supervised learning algorithms with planning methods to solve large, discrete and stochastic RL problems. SPITI is an instance of SDYNA that uses an incremental decision tree induction algorithm combined with an incremental version of SVI. Consequently, SPITI is not able to exploit additive structure of FMDPs and is very limited to represent and solve problems such as the SysAdmin problem [5].

In this paper, we describe two new instances of SDYNA, namely ULP and UNATLP. Both instances use LP based planning method to exploit the additive structure of the problem. Moreover, we propose in UNATLP a different representation of the transition function in the FMDP that is learned. First, we show that both ULP and UNATLP are able to exploit the additive structure of a problem without knowing its structure in advance, allowing these instances to address a RL problem with $4 \cdot 10^{13}$ state/action pairs. At this time, we are not aware of any model-based RL algorithm able to solve such large problems without assuming the knowledge of its structure. Second, we show that the new representation of the FMDP in UNATLP outperforms the one used in SPITI and ULP on such problems.

The remainder of this paper is organized as follows: in Section 2, we introduce FMDPs, the planning method based on LP proposed by [5] and SDYNA. In Section 3, we describe ULP and UNATLP. Section 4 describes and discusses empirical results of these instances on the SysAdmin problem.

2 Background

We first introduce some definitions used in this paper. A MDP is defined by a tuple $\langle S, A, R, P \rangle$ where S is a finite set of states, A is a finite set of actions, R is the immediate *reward function* with $R : S \times A \rightarrow \mathbb{R}$ and P is the *Markovian transition function* $P(s'|s,a)$ with $P : S \times A \times S \rightarrow [0,1]$. A *stationary policy* π is a mapping $S \rightarrow A$ with $\pi(s)$ defining the action to be taken in state s.

We evaluate a policy π in state s, considering an infinite horizon, with the *value function* $V_\pi(s)$ defined using the discounted reward criterion: $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s]$, with $0 \leq \gamma < 1$ the discount factor and r_t the reward obtained at time t. A policy π is optimal if $\forall s \in S, \forall \pi' : V_\pi(s) > V_{\pi'}(s)$. The value function of an optimal policy π^* is called *optimal value function* and is noted V^*.

The action-value function $Q_a^V(s)$ for an action a and a value function $V(s)$ is defined as $Q_a^V(s) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$. For a given value function V, it is possible to define a greedy policy relative to V, noted Greedy_V, by taking for each state s the action with the best action-value:

$$\text{Greedy}_V(s) = \arg\max_a [R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')] \quad (1)$$

The greedy policy relative to V^* is an optimal policy $\pi^*(s) = \text{Greedy}_{V^*}(s)$.

We now assume that states are composed of a set of random variables $X = \{X_1, \ldots, X_n\}$. A state is then defined by a vector $s = (x_1, \ldots, x_n)$ with $\forall i, x_i \in \text{Dom}(X_i)$. FMDPs are a framework exploiting the structure of the problem to represent compactly large MDPs [1]. For each action a, the transition model of the FMDP is defined by a separate DBN model $T_a = \langle G_a, \{P^a_{X_1}, \ldots, P^a_{X_n}\}\rangle$. G_a is a two-layer directed acyclic graph whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ with X_i a variable at time t and X'_i the same variable at time $t+1$. The parents of X'_i are noted $\text{Parents}_a(X'_i)$ with $\text{Parents}_a(X'_i) \subseteq X$. The transition model T_a is quantified by *Conditional Probability Distributions* (CPDs), noted $P^a_{X_i}(X'_i|\text{Parents}_a(X'_i))$, associated to each node $X'_i \in G_a$.

A similar decomposition provides a compact representation of the reward function. First, we formalize the concept of *localized* function [5]: a function f has a *scope* $\text{Scope}(f) = Y \subseteq X$ if $f : \text{Dom}(Y) \mapsto \mathbb{R}$. We use $f(x)$ as shorthand for $f(y)$ where y is the part of the instantiation x corresponding to variables in Y. The reward function may now be defined as the sum $\sum_{j=1}^{r} R_j(s) \in \mathbb{R}$ where each function R_j is a localized function with $\text{Scope}(R_j)$ restricted to a small set of variables. There may be a different decomposition R^a_j for each action a.

2.1 Linear Programming Based Approximation in MDPs

Different approaches may be used to compute the optimal policy in a MDP. One approach is to represent the MDP as a linear program (LP) [7]. Such LP is defined as:

For variables: $V(s), \forall s \in S$
Minimize: $\sum_s \alpha(s)V(s)$
Subject to: $V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \forall s \in S, \forall a \in A$ (LP 1)

where $\alpha(s)$ are the state relevance weights ($\alpha(s) > 0$ for each state s). One major issue that prevents this LP from being applied to real problems is that it uses explicit enumerations of the state space which is usually very large. The remaining of this section presents previous work from [5] to avoid such enumerations.

One approach to address large state space is to approximate value functions with *linear value function* $\sum_{i=1}^{k} w_i h_i(s)$ for some coefficients $(w_1, \ldots, w_k)$. The set $\{h_1, \ldots, h_k\}$ is a set of *basis functions* where each h_i is a localized function, defining the space $\mathcal{H}$ of allowable value functions. We discuss the choice of such functions in section 5. Replacing explicit state value function $V(s)$ by the approximation, (LP 1) is rewritten to produce an approximation of the optimal value function of the MDP in $\mathcal{H}$ [8]:

For variables: $w_1, \ldots, w_k$
Minimize: $\sum_s \alpha(s) \sum_{i=1}^{k} w_i h_i(s)$
Subject to: $\sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) +$
$\gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^{k} w_i h_i(s') \forall s \in S, \forall a \in A$ (LP 2)

This linear program is guaranteed to be feasible if a constant function h_0 is included in the set of basis functions. We assume that this is the case in the remaining of the paper.

(LP 2) reduces the number of free variables from $|S|$ to k. However, the number of constraints remains $|S|\times|A|$, each constraint is potentially a sum of $|S|$ terms, and the objective function is still a sum of $|S|$ terms. We now describe an outline of the method proposed by [5] to represent this linear program compactly, exploiting the structure of the problem.

2.2 Linear Programming Based Approximation in Factored MDPs

The compact representation of (LP 2) is based on a *factored* linear value function representation [9], that is a linear value function over the basis $h_1, \ldots, h_k$ where each localized function h_i is restricted to a small number of state variables. The different restricted domain functions defined in the FMDP result in efficient computations on compact representations over large state spaces. The first operation is to compute the backprojection $g_i^a(s) = \sum_{s'} P(s'|s,a)h_i(s')$ of a basis function h_i for an action a and given the graph G_a. Recalling that the scope of a basis function h_i is small, the backprojection may be simplified, as shown in [9]. The scope of $g_i^a(s)$ is $\text{Scope}(g_i^a) = \cup_{X'_j \in \text{Scope}(h_i)} \text{Parents}_a(X'_j)$. Consequently, the cost of the computation depends linearly on $|\text{Dom}(\text{Scope}(g_i^a))|$, which depends on the scope of h_i and the complexity of the graph G_a.

The state relevance weights $\alpha(s)$ in the objective function of (LP 2) may be considered as distribution over states, so that $\alpha(s) > 0$ and $\sum_s \alpha(s) = 1$. As suggested by [5], we use uniform state relevance weights defined as $\alpha(s) = \frac{1}{|S|}$. Using a different reorganisation of the set of constraints and the objective function, a new linear program may be formulated [5]:

For variables: $w_1, \ldots, w_k$
Minimize: $\sum_{i=1}^{k} w_i \alpha_i$
Subject to: $0 \geq \sum_{i=1}^{k} w_i[\gamma g_i^a(s) - h_i(s)] + \sum_{j=1}^{r} R_j^a(s) \; \forall s \in S, \forall a \in A$ (LP 3)

where $\alpha_i = \sum_{c_i \in \text{Dom}(C_i)} \alpha(c_i)h_i(c_i)$ with $C_i = \text{Scope}(h_i)$ and $\alpha(c_i)$ the marginal of the state relevance weights α over C_i.

(LP 3) has now only k free variables, k terms in the objective function and each constraint may be computed on a restricted scope. However, the number of constraints is still exponential. As described by [5], these constraints may be represented compactly using a variable elimination algorithm. We note FactoredALP the algorithm building (LP 3) and returning the solution. We refer to [5] for a comprehensive description of this algorithm.

2.3 Context-Specific Independence

Similarly to solution methods in FMDP such as SPI, SVI and SPUDD [3,4], [5] exploits context-specific independence, using a rule-based representation [10], to compute the backprojections and to reduce the number of constraints in (LP 3). Entries with the same value of a function are referred to as *consistent* contexts. Two contexts $c \in \text{Dom}(C)$ and $b \in \text{Dom}(B)$ with $C \subseteq \{X, X'\}$ and $B \subseteq \{X, X'\}$ are defined as consistent if they have the same assignment for the variables in $C \cap B$.

A *probability rule* $\eta = |c : p|$ is a function $\eta : \{X, X'_i\} \mapsto [0,1]$, where the context $c \in \mathrm{Dom}(C)$, $C \subseteq \{X, X'_i\}$ and $p \in [0,1]$ and such that $\eta(x, x'_i) = p$ if x and x'_i are consistent with c and $\eta(x, x'_i) = 1$ otherwise. A *rule-based CPD* is a function $P_a : (\{X'_i\} \cup X) \mapsto [0,1]$ composed of a set of probability rules $\{\eta_1, \ldots, \eta_n\}$ whose contexts are mutually exclusive and exhaustive. A *value rule* $\rho = |c : v|$ is a function $\rho : X \to \mathbb{R}$ such that $\rho(s) = v$ when s is consistent with c and 0 otherwise. A *rule-based function* $f : X \mapsto \mathbb{R}$ is composed of a set of rules $\{\rho_1, \ldots, \rho_n\}$ such that $f(x) = \sum_{i=1}^{n} \rho_i(x)$. Both reward and basis functions may be represented as a rule-based function.

The transition, reward and value functions in a FMDP may be represented using rule-based representations. [5] utilizes this representation to rewrite the computation of the backprojections and to represent (LP 3) compactly, exploiting context-specific independence. We refer to their paper for a complete description.

2.4 Structured DYNA and Spiti

The solution method based on LP described in the previous section assumes that the structure of the problem is available, which may not be the case in practice. Recently, [6] has proposed *Structured* DYNA (SDYNA), that is a framework able to learn the structure of a FMDP from experience. SDYNA is described in Figure 1, where $\mathsf{Fact}[F]$ represents a factored representation of a function F.

```
Input: Acting, Learn, Plan, Fact Output: $\emptyset$

1. Initialize the FMDP $\mathcal{F}_0$
2. At each time step $t$, with $s$ the current (non-terminal) state, do:
   (a) $a \leftarrow \mathrm{Acting}(s, \{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\})$
   (b) Execute $a$; observe $s'$ and $r$
   (c) $\mathcal{F}_t \leftarrow \mathrm{Learn}(\mathcal{F}_{t-1}, \langle s, a, s', r \rangle)$
   (d) $\{\mathsf{Fact}[V_t], \{\mathsf{Fact}[Q^a_t], \forall a \in A\}\} \leftarrow \mathrm{Plan}(\mathcal{F}_t, \mathsf{Fact}[V_{t-1}])$
```

Fig. 1. The SDYNA algorithm

SDYNA is decomposed in three phases. First, from its current policy, represented as the set $\{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\}$ of action-value function, an action is executed during the acting phase (step 2.a, 2.b and 2.c). Second, from the observation $\langle s, a, s', r \rangle$, the FMDP $\mathcal{F}$ representing a model of the problem is updated during the learning phase (step 2.d). Finally, the set $\{\mathsf{Fact}[Q^a_{t-1}], \forall a \in A\}$ of action-value functions is updated during the planning phase (step 2.e).

SPITI is an instantiation of SDYNA, also presented in [6], that incrementally learns structured representations of the transition and reward functions using an induction of decision tree algorithm, noted $\mathrm{UpdateTree}(\mathsf{Tree}[F], x, y)$ with $\mathsf{Tree}[F]$ the decision tree representation of the function F to update, x the input of F and y its output. First, for the action a of the observation, each CPD

$\mathsf{Tree}[P^a_{X_i}]$ is updated with UpdateTree($\mathsf{Tree}[P^a_{X_i}], (x_1, \ldots, x_n), x'_i$) using the state $s = (x_1, \ldots, x_n)$ as the set of inputs and the value of the variable X_i in s' as output. Second, the reward function $\mathsf{Tree}[R]$ is updated using the current state and action as input and the reward observed as output. SPITI uses χ^2 as its information-theoric metric. Moreover, a decision node in a tree $\mathsf{Tree}[P^a_{X_i}]$ is installed only if the χ^2 value for the variable is above a threshold τ_{χ^2} [11]. SPITI uses an incremental version of the SVI algorithm [3] during the planning phase to update the set $\{\mathsf{Tree}[Q^a_{t-1}], \forall a \in A\}$ of action-value functions. All the functions of a FMDP, that is the CPDs in the transition function, the reward function and the value function, are represented with decision trees in SPITI, as in SVI.

3 Exploiting Additive Structure in SDYNA

SPITI suffers from two strong limitations to be able to exploit the additive structure of a RL problem. First, the reward function is represented with one tree $\mathsf{Tree}[R]$ which is not adapted to represent functions with an additive structure. Second, it plans with an incremental version of SVI, which is not able to exploit the additive structure of a problem and performs very poorly on such problems [3,5]. We address both issues in the remaining of this section by describing two new instances of SDYNA, namely ULP and UNATLP.

3.1 Learning the Structure

To be able to learn additively decomposed reward functions, we assume that the reward received by the agent from the environment is not a single real number $r \in \mathbb{R}$ but a vector $r = (r_1, \ldots, r_r) \in \mathbb{R}^r$ where each r_j is the reward associated to the localized R_j function. Consequently, we assume neither the knowledge of the scope nor the structure of context independencies of the localized functions. Figure 2 shows the Learn($\mathcal{F}, \langle s, a, s', r\rangle$) algorithm for both ULP and UNATLP, adapted from SPITI, to learn additively decomposed reward functions.

The transition function in ULP is updated in the same way as the transition function in SPITI (step 1). However, because the reward r observed by the agent is now decomposed as a vector of reward $r = (r_1, \ldots, r_r)$, a different tree $\mathsf{Tree}[R_j]$ corresponding to a localized function R_j in $R(s) = \sum_{j=1}^{r} R_j(s)$ is updated using the current state as input and the reward r_j in r as output (step 2).

```
Input: a FMDP $\mathcal{F}$, an observation $\langle s, a, s', r\rangle$ Output: $\emptyset$

 1. For all $X_i \in X$:
    In ULP: UpdateTree($\mathsf{Tree}[P^a_{X_i}], \langle (x_1, \ldots, x_n), x'_i\rangle$)
    In UNATLP: UpdateTree($\mathsf{Tree}[P_{X_i}], \langle (x_1, \ldots, x_n), a, x'_i\rangle$)
 2. For all $R_j \in R$:
    UpdateTree($\mathsf{Tree}[R_j], \langle (x_1, \ldots, x_n), a, r_j\rangle$)
```

Fig. 2. The Learn($\mathcal{F}, \langle s, a, s', r\rangle$) algorithm in ULP and UNATLP

Both SPITI and ULP use one tree $\mathsf{Tree}[P^a_{X_i}]$ for each variable and each action to represent the CPD $P^a_{X_i}$ in a FMDP $\mathcal{F}$. It is also possible to represent the transition function with the action as a variable [12] with only one CPD P_{X_i} for each variable X_i of the problem to quantify only one graph G (with an action node) specifying variable dependencies in the transition function.

One drawback of this representation is that it is not possible to specify a dependency between two variables for only one action in the graph G. For instance, it is not possible to specify that X'_i depends on X_j for the action a_k, but not for the other actions of the MDP. However, this drawback may be counterbalanced by using structured representation of the CPDs to exploit context specific independence because probability distributions that do not depend on the action executed by the agent can be aggregated.

UNATLP uses such representation of the transition function. The difference between ULP and UNATLP to learn the FMDP is in step 1 (Figure 2) where each CPD $\mathsf{Tree}[P_{X_i}]$, that is the tree representing P_{X_i} with the action considered as a variable, is updated for each variable X_i using the current state and the action executed by the agent as the set of attributes and the value of the variable X_i in s' as input.

3.2 Acting and Planning

SDYNA does not need an explicit representation of the policy for the agent to act. However, it requires a set of action-value functions to select an action with the best action-value. We note $\mathsf{Rules}[F]$ a function F represented as a rule-based function. Recalling equation 1 and using the factored linear value function $\mathsf{Rules}[\mathcal{V}]$, we can obtain the best action by computing $\text{Greedy}_{\mathcal{V}}(s) = \arg\max_a [\sum_{j=1}^{r} \mathsf{Rules}[r^a_j](s) + \gamma \sum_{i=1}^{k} w_i \mathsf{Rules}[g^a_i](s)]$. Both ULP and UNATLP instances use $\text{Greedy}_{\mathcal{V}}(s)$ combined with ϵ-*greedy* as exploration policy, which executes the best action most of the time, and, with a small probability ϵ, selects uniformly at random an action. When different actions are considered as best, one of them is selected uniformly at random.

The LP based method, as described in section 2.1, uses rule-based representations to exploit context-specific structure whereas both ULP and UNATLP use tree representations of CPDs. Nevertheless, from $\mathsf{Tree}[P](X'|s)$, we can build the corresponding rule-based representation $\mathsf{Rules}[P](X'|s)$ by composing the set of probability rules such that: $\mathsf{Rules}[P](X'|s) = \{|c_i \wedge X' = x : p_i| \text{ such that } x \in \text{Dom}(X), p_i(X' = x|s) \neq 0, \forall l_i \in \mathsf{Tree}[P](X'|s)\}$ with c_i the context of the leaf l_i and p_i the probability $P(X' = x|s)$ in l_i. A similar conversion is used to obtain rule-based value functions from decision trees.

By using such conversions, the FMDP incrementally learned may be used with linear programming based planning method. We propose, in Figure 3, an incremental planning algorithm able to exploit the additive structure of a problem for both ULP and UNATLP.

The main idea is to re-use the previous solution of the LP when it is available and to avoid to solve the full LP at each time step. First, the $\text{Plan}(\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}])$ algorithm checks if the structure of the FMDP has changed (such information is

```
Input: $\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}]$  Output: $\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$,  Parameters: $T_P, T_M, T_{MIN}$

1. If (Structure of $\mathcal{F}_t \neq$ Structure of $\mathcal{F}_{t-1}$) then:
   (a) lastModif $\leftarrow t$
   (b) Reinitialize the solution of the last linear program
2. If (($t-$lastModif $> T_M$) or ($t-$lastPlanning $< T_P$)) and ($t-$lastPlanning $> T_{MIN}$) then:
   (a) lastPlanning $\leftarrow t$
   (b) $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow$ FactoredALP($\mathcal{F}_t$) using the solution of the last linear program if it has not been reinitialized.
   else: $\{\mathsf{Rules}[\mathcal{V}_t], \{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}\} \leftarrow \{\mathsf{Rules}[\mathcal{V}_{t-1}], \{\mathsf{Rules}[Q_a^{\mathcal{V}_{t-1}}], \forall a \in A\}\}$
3. Return $\mathsf{Rules}[\mathcal{V}_t]$ and $\{\mathsf{Rules}[Q_a^{\mathcal{V}_t}], \forall a \in A\}$
```

Fig. 3. The Plan($\mathcal{F}_t, \mathsf{Rules}[\mathcal{V}_{t-1}]$) algorithm in ULP and UNATLP instances of SDYNA

maintained by the UpdateTree algorithm for each tree of the FMDP). When the structure does not change, then the constraints of the LP have the same structure using the same free variables. Consequently, we use the solution of the last linear program as a feasible solution (step 2b).

When the structure has changed, then we cannot re-use the solution of the last linear program which is reinitialized (step 1). Then, during step 2, if the structure has not changed for T_M time step, or if the last time a linear program has been solved is under T_P and over T_{MIN}, then the current solution is updated. When the solution of the linear program has not been updated, then the algorithm returns the last computed solution.

Note that to use the representation of the transition function of UNATLP, an additional step is required: it is necessary to build the CPD $\mathsf{Tree}[P_{X_i}^a]$ for each action a from the CPD $\mathsf{Tree}[P_{X_i}]$. If the action is not tested in $\mathsf{Tree}[P_{X_i}]$, then $\mathsf{Tree}[P_{X_i}^a] = \mathsf{Tree}[P_{X_i}]$ for all a, else $\mathsf{Tree}[P_{X_i}^a]$ is extracted from $\mathsf{Tree}[P_{X_i}]$ by replacing each decision node testing the action in $\mathsf{Tree}[P_{X_i}]$ by the sub-tree corresponding to a.

4 Results

We now present an empirical evaluation of both ULP and UNATLP. We use the SysAdmin benchmark problem, strictly following the specification given in [5] using the unidirectional ring network architecture with $N = 40$ machines. This problem exhibits a very symmetric model and has been used by [5] to show the performance of FactoredALP. But, unlike the results presented below, the structure of the model was assumed to be fully known. Moreover, [3,4,5] show that SPITI is unsuitable for this problem because of the decision trees used to represent the reward and value functions.

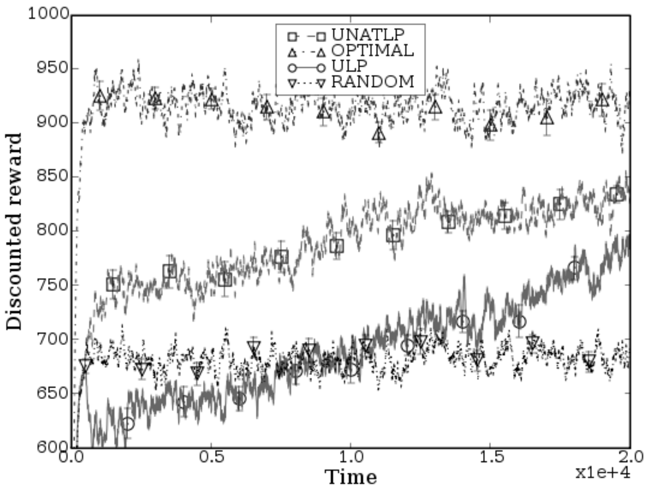


Fig. 4. Discounted reward obtained on the SysAdmin problem. Both ULP and UNATLP are able to improve their policy, despite the large size of the problem. Error bars are present, but hardly visible because of a very low standard deviation.

The size of the problem is $2^{40} \times 41 \approx 4 \cdot 10^{13}$ state/action pairs. A each time step, a SysAdmin may go to one machine to reboot it, making it work for the next time step with a high probability. The SysAdmin receives a reward of 1 for each running machine (except for one machine for which it receives 2 to introduce an asymmetry in the problem). We use N basis functions h_i corresponding to each machine represented by the X_i variable and defined such as $\{|X_{i-1} = 0 \wedge X_i = 0 : 0.05|, |X_{i-1} = 1 \wedge X_i = 0 : 0.09|, |X_{i-1} = 0 \wedge X_i = 1 : 0.5|, |X_{i-1} = 1 \wedge X_i = 1 : 0.9|\}$ and a constant basis function.

We use `glpsol`[1] as LP solver, $\epsilon = 0.1$ in ϵ-*greedy* exploration policy, a discount factor $\gamma = 0.99$, $\tau_{\chi^2} = 30$ in the UpdateTree induction tree algorithm and $T_M = 100$, $T_P = 1500$ and $T_{MIN} = 50$ in the Plan algorithm (Figure 3). We ran 10 experiments of 20,000 time steps. We use the tree induction algorithm ID4 [13]. For implementation reasons, the backprojections $g_i^a(s)$ are computed using decision trees as representations [3]. We refer to the respective papers for respective descriptions. Moreover, we use the least-square criterion [14] to learn reward functions in the UpdateTree algorithm. We ran two more agents, noted RANDOM and OPTIMAL, executing at each time step, respectively, a random action and the best action. The policy of OPTIMAL has been computed off-line, using FactoredALP with the same basis functions as defined below.

Figure 4 shows the discounted reward, defined as $R_t^{disc} = r_t + \gamma R_{t-1}^{disc}$ with r_t the reward received by the agent, obtained by each agent over the time (in time steps). Both ULP and UNATLP are able to substantially improve their policy, compared to RANDOM. Moreover, UNATLP improves fairly quickly its policy compared to ULP and considering the size and the stochasticity of the problem.

[1] http://www.gnu.org/software/glpk/glpk.html

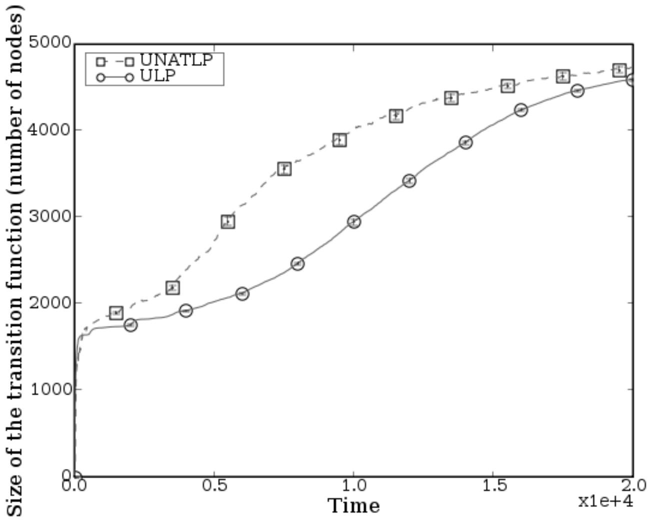


Fig. 5. Size of the transition function of the FMDP learned in ULP and UNATLP. Erros bars are present, but hardly visible because of a very low standard deviation.

Figure 5 compares the different size of the transition function (in nodes, including decision nodes and leaves) learned by ULP and UNATLP. Both build a transition function with a similar size of approximately 4500 nodes (compared to the size of $41 \times 40 \times 7 = 11480$ nodes of the full transition function with perfect knowledge using the ULP representation). We observe that, for a similar size, UNATLP obtains better results than ULP. Moreover, the transition function built by UNATLP grows quicker than the transition function in ULP.

5 Discussion

These results show that ULP and UNATLP are able to quickly learn an accurate FMDP representing the RL problem to solve with few assumptions on this problem. They are able to exploit the additive structure of a problem even when its structure is not known in advance. Thus, by combining LP based planning methods with supervised learning methods such as decision tree induction, ULP and UNATLP are able to address very large problems by exploiting a strong generalisation property. Note however that the size of the model stabilizes in both cases below the size of the perfect model, which suggests that the perfect representation will not be learned. This is mainly due to the ϵ-*greedy* exploration policy we use, which drives the agent to explore only paths near the greedy policy. However, despite an imperfect representation, these results clearly show that the policy is still improved. Including better exploration policy than ϵ-*greedy* is still work in progress.

Moreover, these results illustrate the difference between representations of the transition function in ULP and UNATLP. Figure 4 shows that UNATLP learns quicker than ULP. The main reason is that new examples only update the CPD of the last action executed in ULP, whereas they update all the trees of the transition function in UNATLP. [12] suggests that the second representation may be more compact when the value of a variable persists for all actions. However, such a property could not be observed in this problem because each variable depends on one action. Despite the fact that the ULP representation is usually used in FMDP planning algorithms, these results suggest that the representation used in UNATLP is an interesting alternative to solve RL problems.

Future works include a method for building automatically a good set of basis functions, such as the work of [15]. To our knowledge, ULP and UNATLP are the first algorithms to learn and to exploit a FMDP with a factored transition function and an additively decomposed reward function. Such FMDP can directly be used to construct a set of localized basis functions adapted to FMDP planning methods. Thus, such approach would be very promising, combining the generality of the representations used in ULP and UNATLP with the automation of SPITI to solve large RL problems.

Our first contribution in this paper was to show that LP planning methods may be combined with decision tree induction to address very large RL problems, exploiting additive structure, even when the structure is unknown. Our second contribution is to show that a different representation of the transition function in FMDPs may speed up the learning process, particularly in large RL problems, with no loss on the size of the transition function. To conclude, approaches such as ULP and UNATLP can address RL problems that were out of reach with previous model based methods, as far as no information at all about the structure of the problem is given to the system.

References

1. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting Structure in Policy Construction. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1104–1111 (1995)
2. Dean, T., Kanazawa, K.: A Model for Reasoning about Persistence and Causation. Computational Intelligence 5, 142–150 (1989)
3. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic Dynamic Programming with Factored Representations. Artificial Intelligence 121(1), 49–107 (2000)
4. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: Stochastic Planning using Decision Diagrams. In: Proceedings of the 15th Conference on UAI, pp. 279–288. Morgan Kaufmann, San Francisco (1999)
5. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient Solution Algorithms for Factored MDPs. Journal of Artificial Intelligence Research 19, 399–468 (2003)
6. Degris, T., Sigaud, O., Wuillemin, P.H.: Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. In: Proceedings of the 23rd ICML, pp. 257–264 (2006)
7. Manne, A.S.: Linear Programming and Sequential Decisions. Cowles Foundation for Research in Economics at Yale University (1960)
8. Schweitzer, P., Seidmann, A.: Generalized Polynomial Approximations in Markovian Decision Processes. Journal of Mathematical Analysis and Applications 110, 568–582 (1985)

9. Koller, D., Parr, R.: Computing Factored Value Functions for Policies in Structured MDPs. In: Proceedings 16th International Joint Conference on Artificial Intelligence, pp. 1332–1339 (1999)
10. Zhang, T., Poole, D.: On the Role of Context-specific Independence in Probabilistic Reasoning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence, pp. 1288–1293 (1999)
11. Degris, T., Sigaud, O., Wuillemin, P.H.: Chi-square Tests Driven Method for Learning the Structure of Factored MDPs. In: Proceedings of the 22nd Conference on UAI, pp. 122–129 (2006)
12. Boutilier, C., Goldszmidt, M.: The Frame Problem and Bayesian Network Action Representations. In: Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence, pp. 69–83 (1996)
13. Schlimmer, J., Fisher, D.: A Case Study of Incremental Concept Induction. In: Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 496–501 (1986)
14. Breiman, B., Breiman, L.: Classification and Regression Trees. Chapman & Hall/CRC, Boca Raton (1984)
15. Kveton, B., Hauskrecht, M.: Learning Basis Functions in Hybrid Domains. In: Proceedings of the 21st National Conference on Artificial Intelligence, pp. 1161–1166 (2006)

Algorithms and Bounds for Rollout Sampling Approximate Policy Iteration*

Christos Dimitrakakis[1] and Michail G. Lagoudakis[2]

[1] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
dimitrak@science.uva.nl

[2] Department of ECE, Technical University of Crete, Chania 73100, Greece
lagoudakis@intelligence.tuc.gr

Abstract. Several approximate policy iteration schemes without value functions, which focus on policy representation using classifiers and address policy learning as a supervised learning problem, have been proposed recently. Finding good policies with such methods requires not only an appropriate classifier, but also reliable examples of best actions, covering the state space sufficiently. Up to this time, little work has been done on appropriate covering schemes and on methods for reducing the sample complexity of such methods, especially in continuous state spaces. This paper focuses on the simplest possible covering scheme (a discretized grid over the state space) and performs a sample-complexity comparison between the simplest (and previously commonly used) rollout sampling allocation strategy, which allocates samples equally at each state under consideration, and an almost as simple method, which allocates samples only as needed and requires significantly fewer samples.

1 Introduction

Supervised and reinforcement learning are two well-known learning paradigms, which have been researched mostly independently. Recent studies have investigated using mature supervised learning methods for reinforcement learning [6, 7, 9, 10]. Initial results have shown that policies can be approximately represented using multi-class classifiers and therefore it is possible to incorporate classification algorithms within the inner loops of several reinforcement learning algorithms [6, 7, 9]. This viewpoint allows the quantification of the performance of reinforcement learning algorithms in terms of the performance of classification algorithms [10]. While a variety of promising combinations become possible through this synergy, heretofore there have been limited practical results and widely-applicable algorithms.

Herein we consider approximate policy iteration algorithms, such as those proposed by Lagoudakis and Parr [9] as well as Fern et al. [6, 7], which do not explicitly represent a value function. At each iteration, a new policy/classifier is

* This project was partially supported by the ICIS-IAS project and the Marie Curie International Reintegration Grant MCIRG-CT-2006-044980 awarded to Michail G. Lagoudakis within the 6th European Framework Programme.

produced using training data obtained through extensive simulation (rollouts) of the previous policy on a generative model of the process. These rollouts aim at identifying better action choices over a subset of states in order to form a set of data for training the classifier representing the improved policy. The major limitation of these algorithms, as also indicated by Lagoudakis and Parr [9], is the large amount of rollout sampling employed at each sampled state. It is hinted, however, that great improvement could be achieved with sophisticated management of sampling. We have verified this intuition in a companion paper [4] that experimentally compared the original approach of uninformed uniform sampling with various intelligent sampling techniques. That paper employed heuristic variants of well-known algorithms for bandit problems, such as Upper Confidence Bounds [1] and Successive Elimination [5], for the purpose of managing rollouts (choosing which state to sample from is similar to choosing which lever to pull on a bandit machine). It should be noted, however, that despite the similarity, rollout management has substantial differences to standard bandit problems and thus general bandits results are not directly applicable to our case.

The current paper aims to offer a first theoretical insight into the rollout sampling problem. This is done through the analysis of the two simplest sample allocation methods described in [4]. Firstly, the old method that simply allocates an equal, fixed number of samples at each state and secondly the slightly more sophisticated method of progressively sampling all states where we are not yet reasonably certain of which the policy-improving action would be.

The remainder of the paper is organised as follows. Section 2 provides the necessary background, Section 4 introduces the proposed algorithms, and Section 3 discusses related work. Section 5, which contains an analysis of the proposed algorithms, is the main technical contribution.

2 Preliminaries

A *Markov Decision Process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is a finite set of actions, P is a Markovian transition model ($P(s, a, s')$ denotes the probability of a transition to state s' when taking action a in state s), R is a reward function ($R(s, a)$ is the expected reward for taking action a in state s), $\gamma \in (0, 1]$ is the discount factor for future rewards, and D is the initial state distribution. A *deterministic policy* π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice at state s. The value $V^\pi(s)$ of a state s under a policy π is the expected, total, discounted reward when the process begins in state s and all decisions at all steps are made according to π:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) | s_0 = s, s_t \sim P\right] \quad . \tag{1}$$

The goal of the decision maker is to find an optimal policy π^* that maximises the expected, total, discounted reward from all states; in other words, $V^{\pi^*}(s) \geq V^\pi(s)$ for all policies π and all states $s \in \mathcal{S}$.

Policy iteration (PI) is an efficient method for deriving an optimal policy. It generates a sequence π_1, π_2, ..., π_k of gradually improving policies, which terminates when there is no change in the policy ($\pi_k = \pi_{k-1}$); π_k is an optimal policy. Improvement is achieved by computing V^{π_i} analytically (solving the linear Bellman equations) and the action values

$$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V^{\pi_i}(s') \ ,$$

and then determining the improved policy as $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$.

Policy iteration typically terminates in a small number of steps. However, it relies on knowledge of the full MDP model, exact computation and representation of the value function of each policy, and exact representation of each policy. *Approximate policy iteration* (API) is a family of methods, which have been suggested to address the "curse of dimensionality", that is, the huge growth in complexity as the problem grows. In API, value functions and policies are represented approximately in some compact form, but the iterative improvement process remains the same. Apparently, the guarantees for monotonic improvement, optimality, and convergence are compromised. API may never converge, however in practice it reaches good policies in only a few iterations.

2.1 Rollout Estimates

Typically, API employs some representation of the MDP model to compute the value function and derive the improved policy. On the other hand, the Monte-Carlo estimation technique of *rollouts* provides a way of accurately estimating Q^π at any given state-action pair (s,a) without requiring an explicit MDP model or representation of the value function. Instead, a generative model of the process (a simulator) is used; such a model takes a state-action pair (s,a) and returns a reward r and a next state s' sampled from $R(s,a)$ and $P(s,a,s')$ respectively.

A rollout for the state-action pair (s,a) amounts to simulating a single trajectory of the process beginning from state s, choosing action a for the first step, and choosing actions according to the policy π thereafter up to a certain horizon T. If we denote the sequence of collected rewards during the i-th simulated trajectory as $r_t^{(i)}$, $t = 0,1,2,\dots,T-1$, then the rollout estimate $\hat{Q}_K^{\pi,T}(s,a)$ of the true state-action value function $Q^\pi(s,a)$ is the observed total discounted reward, averaged over all K trajectories:

$$\hat{Q}_K^{\pi,T}(s,a) \triangleq \frac{1}{K}\sum_{i=1}^{K} \tilde{Q}_{(i)}^{\pi,T}(s,a) \ , \qquad \tilde{Q}_{(i)}^{\pi,T}(s,a) \triangleq \sum_{t=0}^{T-1} \gamma^t r_t^{(i)} \ .$$

Similarly, we define $Q^{\pi,T}(s,a) = \mathbf{E}\left(\sum_{t=0}^{T-1}\gamma^{t-1}r_t \middle| a_0{=}a, s_0{=}s, a_t \sim \pi, s_t \sim P\right)$ to be the actual state-action value function up to horizon T. As will be seen later, with a sufficient amount of rollouts and a long horizon T, we can create an improved policy π' from π at any state s, without requiring a model of the MDP.

3 Related Work

Rollout estimates have been used in the Rollout Classification Policy Iteration (RCPI) algorithm [9], which has yielded promising results in several learning domains. However, as stated therein, it is sensitive to the distribution of training states over the state space. For this reason it is suggested to draw states from the discounted future state distribution of the improved policy. This tricky-to-sample distribution, also used by Fern et al. [7], yields better results. One explanation advanced in those studies is the reduction of the potential mismatch between the training and testing distributions of the classifier.

However, in both cases, and irrespectively of the sampling distribution, the main drawback is the excessive computational cost due to the need for lengthy and repeated rollouts to reach a good level of accuracy in the estimation of the value function. In our preliminary experiments with RCPI, it has been observed that most of the effort is spent where the action value differences are either non-existent, or so fine that they require a prohibitive number of rollouts to identify them. In this paper, we propose and analyse sampling methods to remove this performance bottle-neck. By restricting the sampling distribution to the case of a uniform grid, we compare the fixed allocation algorithm (Fixed) [7, 9], whereby a large fixed amount of rollouts is used for estimating the action values in each training state, to a simple incremental sampling scheme based on counting (Count), where the amount of rollouts in each training state varies. We then derive complexity bounds, which show a clear improvement using Count that depends only on the structure of differential value functions.

We note that Fern et al. [7] presented a related analysis. While they go into considerably more depth with respect to the classifier, their results are not applicable to our framework. This is because they assume that there exists some real number $\Delta^* > 0$ which lower-bounds the amount by which the value of an optimal action(s) under any policy exceeds the value of the nearest sub-optimal action in any state s. Furthermore, the algorithm they analyse uses a fixed number of rollouts at each sampled state. For a given minimum Δ^* value over all states, they derive the necessary number of rollouts per state to guarantee an improvement step with high probability, but the algorithm offers no practical way to guarantee a high probability improvement. We instead derive error bounds for the fixed and counting allocation algorithms. Additionally, we are considering continuous, rather than discrete, state spaces. Because of this, technically our analysis is much more closely related to that of Auer et al. [2].

4 Algorithms to Reduce Sampling Cost

The total sampling cost depends on the balance between the number of states sampled and the number of samples per state. In the fixed allocation scheme [7, 9], the same number of $K|\mathcal{A}|$ rollouts is allocated to each state in a subset S of states and all K rollouts dedicated to a single action are exhausted before moving on to the next action. Intuitively, if the desired outcome (superiority of

Algorithm 1. SampleState

```
Input: state $s$, policy $\pi$, horizon $T$, discount factor $\gamma$
for (each $a \in \mathcal{A}$) do
    $(s', r) = \textsc{Simulate}(s, a)$
    $\tilde{Q}^\pi(s, a) = r$
    $x = s'$
    for $t = 1$ to $T - 1$ do
        $(x', r) = \textsc{Simulate}(x, \pi(x))$
        $\tilde{Q}^\pi(s, a) = \tilde{Q}^\pi(s, a) + \gamma^t r$
        $x = x'$
    end for
end for
return $\tilde{Q}^\pi$
```

some action) in some state can be confidently determined early, there is no need to exhaust all $K|\mathcal{A}|$ rollouts available in that state; the training data could be stored and the state could be removed from the pool without further examination. Similarly, if we can confidently determine that all actions are indifferent in some state, we can simply reject it without wasting any more rollouts; such rejected states could be replaced by fresh ones which might yield meaningful results. These ideas lead to the following question: can we examine all states in S collectively in some interleaved manner by selecting each time a single state to focus on and allocating rollouts only as needed?

Selecting states from the state pool could be viewed as a problem akin to a multi-armed bandit problem, where each state corresponds to an arm. Pulling a lever corresponds to sampling the corresponding state once. By *sampling a state* we mean that we perform a single rollout for each action in that state as shown in Algorithm 1. This is the minimum amount of information we can request from a single state.[1] Thus, the problem is transformed to a *variant* of the classic multi-armed bandit problem. Several methods have been proposed for various versions of this problem, which could potentially be used in this context. In this paper, apart from the fixed allocation scheme presented above, we also examine a simple counting scheme.

The algorithms presented here maintain an empirical estimate $\hat{\Delta}^\pi(s)$ of the marginal difference of the apparently maximal and the second best of actions. This can be represented by the marginal difference in Q^π values in state s, defined as

$$\Delta^\pi(s) = Q^\pi(s, a^*_{s,\pi}) - \max_{a \neq a^*_{s,\pi}} Q^\pi(s, a) \ ,$$

where $a^*_{s,\pi}$ is the action that maximises Q^π in state s:

$$a^*_{s,\pi} = \arg\max_{a \in \mathcal{A}} Q^\pi(s, a) \ .$$

[1] It is possible to also manage sampling of the actions, but herein we are only concerned with the effort saved by managing state sampling.

The case of multiple equivalent maximising actions can be easily handled by generalising to sets of actions in the manner of Fern et al. [7], in particular

$$
\begin{aligned}
A^*_{s,\pi} &\triangleq \{a \in \mathcal{A} : Q^\pi(s,a) \geq Q^\pi(s,a'),\ \forall a' \in \mathcal{A}\} \\
V^\pi_*(s) &= \max_{a \in \mathcal{A}} Q^\pi(s,a) \\
\Delta^\pi(s) &= \begin{cases} V^\pi_*(s) - \max_{a \notin A^*_{s,\pi}} Q^\pi(s,a), & A^*_{s,\pi} \subset \mathcal{A} \\ 0, & A^*_{s,\pi} = \mathcal{A} \end{cases}
\end{aligned}
$$

However, here we discuss only the single best action case to simplify the exposition. The estimate $\hat{\Delta}^\pi(s)$ is defined using the empirical value function $\hat{Q}^\pi(s,a)$.

5 Complexity of Sampling-Based Policy Improvement

Rollout algorithms can be used for policy improvement under certain conditions. Bertsekas [3] gives several theorems for policy iteration using rollouts and an approximate value function that satisfies a consistency property. Specifically, Proposition 3.1. therein states that the one-step look-ahead policy π' computed from the approximate value function $\hat{V}^\pi$, has a value function which is better than the current approximation $\hat{V}^\pi$, if $\max_{a \in \mathcal{A}} \mathbf{E}[r_{t+1} + \gamma \hat{V}^\pi(s_{t+1}) | \pi', s_t = s, a_t = a] \geq \hat{V}^\pi(s)$ for all $s \in \mathcal{S}$. It is easy to see that an approximate value function that uses only sampled trajectories from a fixed policy π satisfies this property if we have an adequate number of samples. While this assures us that we can perform rollouts at any state in order to improve upon the given policy, it does not lend itself directly to policy iteration. That is, with no way to compactly represent the resulting rollout policy we would be limited to performing deeper and deeper tree searches in rollouts.

In this section we shall give conditions that allow policy iteration through compact representation of rollout policies via a grid and a finite number of sampled states and sample trajectories with a finite horizon. Following this, we will analyse the complexity of the fixed sampling allocation scheme employed in [7, 9] and compare it with an oracle that needs only one sample to determine $a^*_{s,\pi}$ for any $s \in \mathcal{S}$ and a simple counting scheme.

5.1 Sufficient Conditions

Assumption 1 (Bounded finite-dimension state space). *The state space $\mathcal{S}$ is a compact subset of* $[0,1]^d$.

This assumption can be generalised to other bounded state spaces easily. However, it is necessary to have this assumption in order to be able to place some minimal constraints on the search.

Assumption 2 (Bounded rewards). $R(s,a) \in [0,1]$ *for all* $a \in \mathcal{A}$, $s \in \mathcal{S}$.

This assumption bounds the reward function and can also be generalised easily to other bounding intervals.

Assumption 3 (Hölder Continuity). *For any policy $\pi \in \Pi$, there exists $L, \alpha \in [0, 1]$, such that for all states $s, s' \in \mathcal{S}$*

$$|Q^\pi(s, a) - Q^\pi(s', a)| \leq \frac{L}{2} \|s - s'\|_\infty^\alpha \ .$$

This assumption ensures that the value function Q^π is fairly smooth. It trivially follows in conjunction with Assumptions 1 and 2 that Q^π, Δ^π are bounded *everywhere* in $\mathcal{S}$ if they are bounded for at least one $s \in \mathcal{S}$. Furthermore, the following holds:

Remark 1. Given that, by definition, $Q^\pi(s, a^*_{s,\pi}) \geq \Delta^\pi(s) + Q^\pi(s, a)$ for all $a \neq a^*_{s,\pi}$, it follows from Assumption 3 that

$$Q^\pi(s', a^*_{s,\pi}) \geq Q^\pi(s', a) \ ,$$

for all $s' \in \mathcal{S}$ such that $\|s - s'\|_\infty \leq \sqrt[\alpha]{\Delta^\pi(s)/L}$.

This remark implies that the best action in some state s according to Q^π will also be the best action in a neighbourhood of states around s. This is a reasonable condition as there would be no chance of obtaining a reasonable estimate of the best action in any region from a single point, if Q^π could change arbitrarily fast. We assert that MDPs with a similar smoothness property on their transition distribution will also satisfy this assumption.

Finally, we need an assumption that limits the total number of rollouts that we need to take, as states with a smaller Δ^π will need more rollouts.

Assumption 4 (Measure). *If $\mu\{S\}$ denotes the Lebesgue measure of set S, then, for any $\pi \in \Pi$, there exist $M, \beta > 0$ such that $\mu\{s \in \mathcal{S} : \Delta^\pi(s) < \epsilon\} < M\epsilon^\beta$ for all $\epsilon > 0$.*

This assumption effectively limits the amount of times value-function changes lead to best-action changes, as well as the ratio of states where the action values are close. This assumption, together with the Hölder continuity assumption, imposes a certain structure on the space of value functions. We are thus guaranteed that the value function of any policy results in an improved policy which is not arbitrarily complex. This in turn, implies that an optimal policy cannot be arbitrarily complex either.

A final difficulty is determining whether there exists some sufficient horizon T_0 beyond which it is unnecessary to go. Unfortunately, even though for any state s for which $Q^\pi(s, a') > Q^\pi(s, a)$, there exists $T_0(s)$ such that $Q^{\pi,T}(s, a') > Q^{\pi,T}(s, a)$ for all $T > T_o(s)$, T_0 grows without bound as we approach a point where the best action changes. However, by selecting a fixed, sufficiently large rollout horizon, we can still behave optimally with respect to the true value function in a compact subset of $\mathcal{S}$.

Lemma 1. *For any policy $\pi \in \Pi$, $\epsilon > 0$, there exists a finite $T_\epsilon > 0$ and a compact subset $\mathcal{S}_\epsilon \subset \mathcal{S}$ such that*

$$Q^{\pi,T}(s, a^*_{s,\pi}) \geq Q^{\pi,T}(s, a) \quad \forall a \in \mathcal{A}, s \in \mathcal{S}, T > T_\epsilon$$

where $a^*_{s,\pi} \in \mathcal{A}$ is such that $Q^\pi(s, a^*_{s,\pi}) \geq Q^\pi(s, a)$ for all $a \in \mathcal{A}$.

Proof. From the above assumptions it follows directly that for any $\epsilon > 0$, there exists a compact set of states $\mathcal{S}_\epsilon \subset \mathcal{S}$ such that $Q^\pi(s, a^*_{s,\pi}) \geq Q^\pi(s, a') + \epsilon$ for all $s \in \mathcal{S}_\epsilon$, with $a' = \arg\max_{a \neq a^*_{s,\pi}} Q^\pi(s, a)$. Now let $x_T \triangleq Q^{\pi,T}(s, a^*_{s,\pi}) - Q^{\pi,T}(s, a')$. Then, $x_\infty \triangleq \lim_{T\to\infty} x_T \geq \epsilon$. For any $s \in \mathcal{S}_\epsilon$ the limit exists and thus by definition $\exists T_\epsilon(s)$ such that $x_{T_\epsilon} > 0$ for all $T > T_\epsilon$. Since $\mathcal{S}_\epsilon$ is compact, $T_\epsilon \triangleq \sup_{s\in\mathcal{S}_\epsilon} T_\epsilon(s)$ also exists.[2] □

This ensures that we can identify the best action within ϵ, using a finite rollout horizon, in most of $\mathcal{S}$. Moreover, $\mu\{\mathcal{S}_\epsilon\} \geq 1 - M2\epsilon^\beta$ from Assumption 4.

In standard policy iteration, the improved policy π' over π has the property that the improved action in any state is the action with the highest Q^π value in that state. However, in rollout-based policy iteration, we may only guarantee being within $\epsilon > 0$ of the maximally improved policy.

Definition 1 (ϵ-improved policy). *An ϵ-improved policy π' derived from π satisfies*

$$\max_{a\in\mathcal{A}} Q^\pi(s, a) - \epsilon \leq V^{\pi'}(s), \tag{2}$$

Such a policy will be said to be *improving in S* if $V^\pi(s) \leq V^{\pi'}(s)$ for all $s \in S$. The measure of states for which there can not be improvement is limited by Assumption 4. Finding an improved π' for the whole of $\mathcal{S}$ is in fact not possible in finite time, since this requires determining the boundaries in $\mathcal{S}$ at which the best action changes. [3]

In all cases, we shall attempt to find the improving action $a^*_{s,\pi}$ at each state s on a uniform grid of n states, with the next policy $\pi'(s')$ taking the estimated best action $\hat{a}^*_{s,\pi}$ for the state s closest to s', i.e. it is a nearest-neighbour classifier.

In the remainder, we derive complexity bounds for achieving an ϵ-improved policy π' from π with probability at least $1 - \delta$. We shall always assume that we are using a sufficiently deep rollout to cover $\mathcal{S}_\epsilon$ and only consider the number of rollouts performed. First, we shall derive the number of states we need to sample from in order to guarantee an ϵ-improved policy, under the assumption that at each state we have an *oracle* which can give us the exact Q^π values for each state we examine. Later, we shall consider sample complexity bounds for the case where we do not have an oracle, but use empirical estimates $\hat{Q}^{\pi,T}$ at each state.

5.2 The Oracle Algorithm

Let $B(s, \rho)$ denote the infinity-norm sphere of radius ρ centred in s and consider Alg. 2 (Oracle) that can instantly obtain the state-action value function for any point in $\mathcal{S}$. The algorithm creates a uniform grid of n states, such that the distance between adjacent states is $2\rho = \frac{1}{n^{1/d}}$ – and so can cover $\mathcal{S}$ with spheres $B(s, \rho)$. Due to Assumption 3, the error in the action values of any state in sphere

[2] For a discount factor $\gamma < 1$ we can simply bound T_ϵ with $\log[\epsilon(1-\gamma)]/\log(\gamma)$.

[3] To see this, consider $\mathcal{S} \triangleq [0, 1]$, with some $s^* : R(s, a_1) \geq R(s, a_2)\ \forall s \geq s^*$ and $R(s, a_1) < R(s, a_2)\ \forall s < s^*$. Finding s^* requires a binary search, at best.

Algorithm 2. ORACLE

```
Input: $n$, $\pi$
Set $S$ to a uniform grid of $n$ states in $\mathcal{S}$.
for $s \in S$ do
    $\hat{a}^*_{s,\pi} = a^*_{s,\pi}$
end for
return $\hat{A}^*_{S,\pi} \triangleq \{\hat{a}^*_{s,\pi} : s \in S\}$
```

$B(s, \rho)$ of state s will be bounded by $L\left(\frac{1}{2n^{1/d}}\right)^{\alpha}$. Thus, the resulting policy will be $L\left(\frac{1}{2n^{1/d}}\right)^{\alpha}$-improved, i.e. this will be the maximum regret it will suffer over the maximally improved policy.

To bound this regret by ϵ, it is sufficient to have $n = \left(\frac{1}{2}\sqrt[\alpha]{\frac{L}{\epsilon}}\right)^d$ states in the grid. The following proposition follows directly.

Proposition 1. *Algorithm 2 results in regret ϵ for $n = \mathcal{O}\left(L^{d/\alpha}\left[2\epsilon^{1/\alpha}\right]^{-d}\right)$.*

Furthermore, as for all s such that $\Delta^\pi(s) > L\rho^\alpha$, $a^*_{s,\pi}$ will be the improved action in all of $B(s, \rho)$, then π' will be improving in S with $\mu\{S\} \geq 1 - ML^\beta\left(\frac{1}{2n^{1/d}}\right)^{\alpha\beta}$. Both the regret and the lack of complete coverage are due to the fact that we cannot estimate the best-action boundaries with arbitrary precision in finite time. When using rollout sampling, however, even if we restrict ourselves to ϵ improvement, we may still make an error due to both the limited number of rollouts and the finite horizon of the trajectories. In the remainder, we shall derives error bounds for two practical algorithms that employ a fixed grid with a finite number of T-horizon rollouts.

5.3 Error Bounds for States

When we estimate the value function at each $s \in S$ using rollouts there is a probability that the estimated best action $\hat{a}^*_{s,\pi}$ is not in fact the best action. For any given state under consideration, we can apply the following well-known lemma to obtain a bound on this error probability.

Lemma 2 (Hoeffding inequality). *Let X be a random variable in $[b, b+Z]$ with $\bar{X} \triangleq \mathbf{E}[X]$, observed values $X_1, \ldots, X_n$ of X, and $\hat{X}_n \triangleq \frac{1}{n}\sum_{i=1}^n X_i$. Then, $\mathbf{P}(\hat{X}_n \geq \bar{X} + \epsilon) = \mathbf{P}(\hat{X}_n \leq \bar{X} + \epsilon) \leq \exp\left(-2n\epsilon^2/Z^2\right)$ for any $\epsilon > 0$.*

Without loss of generality, consider two random variables $X, Y \in [0, 1]$, with empirical means $\hat{X}_n, \hat{Y}_n$ and empirical difference $\hat{\Delta}_n \triangleq \hat{X}_n - \hat{Y}_n > 0$. Their means and difference will be denoted as $\bar{X}, \bar{Y}, \bar{\Delta} \triangleq \bar{X} - \bar{Y}$ respectively.

Note that if $\bar{X} > \bar{Y}$, $\hat{X}_n > \bar{X} - \bar{\Delta}/2$ and $\hat{Y}_n < \bar{Y} + \bar{\Delta}/2$ then necessarily $\hat{X}_n > \hat{Y}_n$, so $\mathbf{P}(\hat{X}_n > \hat{Y}_n | \bar{X} > \bar{Y}) \geq \mathbf{P}(\hat{X}_n > \bar{X} - \bar{\Delta}/2 \wedge \hat{Y}_n < \bar{Y} + \hat{\Delta}_n/2)$. The converse is

$$\mathbf{P}\left(\hat{X}_n < \hat{Y}_n \mid \bar{X} > \bar{Y}\right) \leq \mathbf{P}\left(\hat{X}_n < \bar{X} - \bar{\Delta}/2 \vee \hat{Y}_n > \bar{Y} + \bar{\Delta}/2\right) \qquad (3a)$$

Algorithm 3. FIXED

```
Input: $n$, $\pi$, $c$, $T$, $\delta$
Set $S$ to a uniform grid of $n$ states in $\mathcal{S}$.
for $s \in S$ do
  Estimate $\hat{Q}_c^{\pi,T}(s,a)$ for all $a$.
  if $\hat{\Delta}^\pi(s) > Z\sqrt{\frac{2\log(2n|\mathcal{A}|/\delta)}{c}}$ then
    $\hat{a}^*_{s,\pi} = \arg\max \hat{Q}^\pi$
  else
    $\hat{a}^*_{s,\pi} = \pi(s)$
  end if
end for
return $\hat{A}^*_{S,\pi} \triangleq \{\hat{a}^*_{s,\pi} : s \in S\}$
```

$$\leq \mathbf{P}\left(\hat{X}_n < \bar{X} - \bar{\Delta}/2\right) + \mathbf{P}\left(\hat{Y}_n > \bar{Y} + \bar{\Delta}/2\right) \tag{3b}$$

$$\leq 2\exp\left(-\frac{n}{2}\bar{\Delta}^2\right). \tag{3c}$$

Now, consider $\hat{a}^*_{s,\pi}$ such that $\hat{Q}^\pi(s,\hat{a}^*_{s,\pi}) \geq \hat{Q}^\pi(s,a)$ for all a. Setting $\hat{X}_n = Z^{-1}\hat{Q}^\pi(s,\hat{a}^*_{s,\pi})$ and $\hat{Y}_n = Z^{-1}\hat{Q}^\pi(s,a)$, where Z is a normalising constant such that $Q \in [b, b+1]$, we can apply (3). Note that the bound is largest for the action a' with value closest to $a^*_{s,\pi}$, for which it holds that $Q^\pi(s,a^*_{s,\pi}) - Q^\pi(s,a') = \Delta^\pi(s)$. Using this fact and an application of the union bound, we conclude that for any state s, from which we have taken $c(s)$ samples, it holds that:

$$\mathbf{P}[\exists \hat{a}^*_{s,\pi} \neq a^*_{s,\pi} : \hat{Q}^\pi(s,\hat{a}^*_{s,\pi}) \geq \hat{Q}^\pi(s,a)] \leq 2|\mathcal{A}|\exp\left(-\frac{c(s)}{2Z^2}\Delta^\pi(s)^2\right). \tag{4}$$

5.4 Uniform Sampling: The FIXED Algorithm

As we have seen in the previous section, if we employ a grid of n states, covering $\mathcal{S}$ with spheres $B(s,\rho)$, where $\rho = \frac{1}{2n^{1/d}}$, and taking action $a^*_{s,\pi}$ in each sphere centred in s, then the resulting policy π' is only guaranteed to be improved within ϵ of the optimal improvement from π, where $\epsilon = L\rho^\alpha$. Now, we examine the case where, instead of obtaining the true $a^*_{s,\pi}$, we have an estimate $\hat{a}^*_{s,\pi}$ arising from c samples from each action in each state, for a total of $cn|\mathcal{A}|$ samples. Algorithm 3 accepts (i.e. it sets $\hat{a}^*_{s,\pi}$ to be the empirically highest value action in that state) for all states satisfying:

$$\hat{\Delta}^\pi(s) \geq Z\sqrt{\frac{2\log(2n|\mathcal{A}|/\delta)}{c}}. \tag{5}$$

The condition ensures that the probability that $Q^\pi(s,\hat{a}^*_{s,\pi}) < Q^\pi(s,a^*_{s,\pi})$, meaning the optimally improving action is not $\hat{a}^*_{s,\pi}$, at any state is at most δ. This can easily be seen by substituting the right hand side of (5) for ϵ in (4). As $\Delta^\pi(s) > 0$, this results in an error probability of a single state smaller than δ/n and we can use a union bound to obtain an error probability of δ for each policy improvement step.

For each state $s \in S$ that the algorithm considers, the following two cases are of interest: (a) $\Delta^\pi(s) < \epsilon$, meaning that even when we have correctly identified $a^*_{s,\pi}$, we are still not improving over all of $B(s,\rho)$ and (b) $\Delta^\pi(s) \geq \epsilon$.

While the probability of accepting the wrong action is always bounded by δ, we must also calculate the probability that we fail to accept an action at all, when $\Delta^\pi(s) \geq \epsilon$ to estimate the expected regret. Restating our acceptance condition as $\hat{\Delta}^\pi(s) \geq \theta$, this is given by:

$$\begin{aligned}\mathbf{P}[\hat{\Delta}^\pi(s) < \theta] &= \mathbf{P}[\hat{\Delta}^\pi(s) - \Delta^\pi(s) < \theta - \Delta^\pi(s)] \\ &= \mathbf{P}[\Delta^\pi(s) - \hat{\Delta}^\pi(s) > \Delta^\pi(s) - \theta], \quad \Delta^\pi(s) > \theta. \end{aligned} \tag{6}$$

Is $\Delta^\pi(s) > \theta$? Note that for $\Delta^\pi(s) > \epsilon$, if $\epsilon > \theta$ then so is Δ^π. So, in order to achieve total probability δ for all state-action pairs in this case, after some calculations, we arrive at this expression for the regret

$$\epsilon = \max\left\{L\left(\frac{1}{2n^{1/d}}\right)^\alpha, Z\sqrt{\frac{8\log(2n|\mathcal{A}|/\delta)}{c}}\right\}. \tag{7}$$

By equating the two sides, we get an expression for the minimum number of samples necessary per state:

$$c = 8\frac{Z^2}{L^2}4^\alpha n^{2\alpha/d}\log(2n|\mathcal{A}|/\delta).$$

This directly allows us to state the following proposition.

Proposition 2. *The sample complexity of Algorithm 3 to achieve regret at most ϵ with probability at least $1-\delta$ is $\mathcal{O}\left(\epsilon^{-2}L^{d/\alpha}\left[2\epsilon^{1/\alpha}\right]^{-d}\log\frac{2|\mathcal{A}|}{\delta}L^{d/\alpha}\left[2\epsilon^{1/\alpha}\right]^{-d}\right)$.*

5.5 The Count Algorithm

The Count algorithm starts with a policy π and a set of states S_0, with $n = |S_0|$. At each iteration k, each sample in S_k is sampled once. Once a state $s \in S_k$ contains a dominating action, it is removed from the search. So,

$$S_k = \left\{s \in S_{k-1} : \hat{\Delta}^\pi(s) < Z\sqrt{\frac{\log(2n|\mathcal{A}|/\delta)}{c(s)}}\right\}$$

Thus, the number of samples from each state is $c(s) \geq k$ if $s \in S_k$.

We can apply similar arguments to analyse Count, by noting that the algorithm spends less time in states with higher Δ^π values. The measure assumption

Algorithm 4. COUNT

```
Input: $n$, $\pi$, $C$, $T$, $\delta$
Set $S_0$ to a uniform grid of $n$ states in $\mathcal{S}$, $c_1, \ldots, c_n = 0$.
for $k = 1, 2, \ldots$ do
  for $s \in S_k$ do
    Estimate $\hat{Q}_c^{\pi,T}(s, a)$ for all $a$, increment $c(s)$
    $S_k = \left\{s \in S_{k-1} : \hat{\Delta}^\pi(s) < Z\sqrt{\frac{2\log(2n|\mathcal{A}|/\delta)}{c(s)}}\right\}$
  end for
  if $\sum_s c(s) >= C$ then
    Break.
  end if
end for
```

then allows us to calculate the number of states with large Δ^π and thus, the number of samples that are needed.

We have already established that there is an upper bound on the regret depending on the grid resolution $\epsilon < L\rho^\alpha$. We proceed by forming subsets of states $W_m = \{s \in S : \Delta^\pi(s) \in [2^{-m}, 2^{1-m})\}$. Note that we only need to consider $m < 1 + \frac{1}{\log 1/2}(\log L + \alpha \log \rho)$.

Similarly to the previous algorithm, and due to our acceptance condition, for each state $s \in W_m$, we need $c(s) \geq 2^{2m+1} Z^2 \log \frac{2n|\mathcal{A}|}{\delta}$ in order to bound the total error probability by δ. The total number of samples necessary is

$$Z^2 \log \frac{2n|\mathcal{A}|}{\delta} \sum_{m=0}^{\lceil \frac{1}{\log 1/2}(\log L + \alpha \log \rho)\rceil} |W_m| 2^{2m+1}.$$

A bound on $|W_m|$ is required to bound this expression. Note that

$$\mu\left\{B(s,\rho) : \Delta^\pi(s') < \epsilon \forall s' \in B(s,\rho)\right\} \leq \mu\left\{s : \Delta^\pi(s) < \epsilon\right\} < M\epsilon^\beta. \tag{8}$$

It follows that $|W_m| < M 2^{\beta(1-m)} \rho^{-d}$ and consequently

$$\begin{aligned}\sum_{s \in S} c(s) &= Z^2 \log \frac{2n|\mathcal{A}|}{\delta} \sum_{m=0}^{\lceil \frac{1}{\log 1/2}(\log L + \alpha \log \rho)\rceil} M 2^{\beta(1-m)} \rho^{-d} 2^{2m+1} \\ &\leq M 2^{\beta+1} 2^{\frac{1+\frac{1}{\log 1/2}(\log L + \alpha \log \rho)}{2-\beta}} 2^d Z^2 n \log \frac{2n|\mathcal{A}|}{\delta}.\end{aligned} \tag{9}$$

The above results directly in the following proposition:

Proposition 3. *The sample complexity of Algorithm 4 to achieve regret at most ϵ with probability at least $1 - \delta$, is $\mathcal{O}\left(L^{d/\alpha}\left[2\epsilon^{1/\alpha}\right]^{-d} \log \frac{2|\mathcal{A}|}{\delta} L^{d/\alpha}\left[2\epsilon^{1/\alpha}\right]^{-d}\right)$.*

We note that we are of course not able to remove the dependency on d, which is only due to the use of a grid. Nevertheless, we obtain a reduction in sample complexity of order ϵ^{-2} for this very simple algorithm.

6 Discussion

We have derived performance pounds for approximate policy improvement without a value function in continuous MDPs. We compared the usual approach of sampling equally from a set of candidate states to the slightly more sophisticated method of sampling from all candidate states in parallel, and removing a candidate state from the set as soon as it was clear which action is best. For the second algorithm, we find an improvement of approximately ϵ^{-2}. Our results complement those of Fern et al [7] for relational Markov decision processes. However significant amount of future work remains.

Firstly, we have assumed everywhere that $T > T_\epsilon$. While this may be a relatively mild assumption for $\gamma < 1$, it is problematic for the undiscounted case, as some states would require far deeper rollouts than others to achieve regret ϵ. Thus, in future work we would like to examine sample complexity in terms of the depth of rollouts as well.

Secondly, we would like to extend the algorithms to increase the number of states that we look at: whenever $\hat{V}^\pi(s) \approx \hat{V}^{\pi'}(s)$ for all s, then we could increase the resolution. For example if,

$$\sum_{s \in S} \mathbf{P}\left(\hat{V}^\pi(s) + \epsilon < \hat{V}^{\pi'}(s) \mid V^\pi(s) > V^{\pi'}(s)\right) < \delta$$

then we could increase the resolution around those states with the smallest Δ^π. This would get around the problem of having to select n.

A related point that has not been addressed herein, is the choice of policy representation. The grid-based representation probably makes poor use of the available number of states. For the increased-resolution scheme outlined above, a classifier such as k-nearest-neighbour could be employed. Furthermore, regularised classifiers might affect a smoothing property on the resulting policy, and allow the learning of improved policies from a set of states containing erroneous best action choices.

As far as the state allocation algorithms are concerned, in a companion paper [4], we have compared the performance of COUNT and FIXED with additional allocation schemes inspired from the UCB and successive elimination algorithms. We have found that all methods outperform FIXED in practice, sometimes by an order of magnitude, with the UCB variants being the best overall.

For this reason, in future work we plan to perform an analysis of such algorithms. A further extension to deeper searches, by for example managing the sampling of actions within a state, could also be performed using techniques similar to [8].

Acknowledgements. Thanks to the reviewers and to Adam Atkinson, Brendan Barnwell, Frans Oliehoek, Ronald Ortner and D. Jacob Wildstrom for comments and useful discussions.

References

[1] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning Journal 47(2-3), 235–256 (2002)
[2] Auer, P., Ortner, R., Szepesvari, C.: Improved Rates for the Stochastic Continuum-Armed Bandit Problem. In: Bshouty, N.H., Gentile, C. (eds.) COLT 2007. LNCS, vol. 4539, pp. 454–468. Springer, Heidelberg (2007)
[3] Bertsekas, D.: Dynamic programming and suboptimal control: From ADP to MPC. Fundamental Issues in Control, European Journal of Control 11(4-5) (2005); From 2005 CDC, Seville, Spain
[4] Dimitrakakis, C., Lagoudakis, M.: Rollout sampling approximate policy iteration. Machine Learning 72(3) (September 2008)
[5] Even-Dar, E., Mannor, S., Mansour, Y.: Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. Journal of Machine Learning Research 7, 1079–1105 (2006)
[6] Fern, A., Yoon, S., Givan, R.: Approximate policy iteration with a policy language bias. Advances in Neural Information Processing Systems 16(3) (2004)
[7] Fern, A., Yoon, S., Givan, R.: Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. Journal of Artificial Intelligence Research 25, 75–118 (2006)
[8] Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS, vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
[9] Lagoudakis, M.G., Parr, R.: Reinforcement learning as classification: Leveraging modern classifiers. In: Proceedings of the 20th International Conference on Machine Learning (ICML), Washington, DC, USA, pp. 424–431 (August 2003)
[10] Langford, J., Zadrozny, B.: Relating reinforcement learning performance to classification performance. In: Proceedings of the 22nd International Conference on Machine learning (ICML), Bonn, Germany, pp. 473–480 (2005)

Efficient Reinforcement Learning in Parameterized Models: Discrete Parameter Case

Kirill Dyagilev[1], Shie Mannor[2], and Nahum Shimkin[1]

[1] Department of EE, Technion, Haifa, Israel
{kirilld@tx,shimkin@ee}.technion.ac.il
[2] Department of ECE, McGill University, Montreal, Canada
shie.mannor@mcgill.ca

Abstract. We consider reinforcement learning in the parameterized setup, where the model is known to belong to a finite set of Markov Decision Processes (MDPs) under the discounted return criteria. We propose an on-line algorithm for learning in such parameterized models, the Parameter Elimination (PEL) algorithm, and analyze its performance in terms of the total mistake bound criterion. The algorithm relies on Wald's sequential probability ratio test to eliminate unlikely parameters, and uses an optimistic policy for effective exploration. We establish that, with high probability, the total mistake bound for the algorithm is linear (up to a logarithmic term) in the size $|\Theta|$ of the parameter space, independently of the cardinality of the state and action spaces. We further demonstrate that much better dependence on $|\Theta|$ is possible, depending on the specific information structure of the problem.

1 Introduction

In Reinforcement Learning (RL), an agent interacts with a partially known environment with purpose of maximizing some numerical utility measure, based on observations of the environment state and reward signals [13]. The environment is often modeled as a Markov Decision Process (MDP) with finite state and action spaces. One possible goal for the agent is to learn an (almost-)optimal control policy as quickly as possible. Alternatively, in an on-line setting where learning is performed during normal system operation, the agent's goal may be to maximize the actually obtained reward or to minimize the number of suboptimal moves relative to the optimal policy.

A fundamental issue that greatly affects the convergence rate of RL algorithms is the efficiency of exploration of the state and action spaces. In an on-line setting, in particular, the agent faces the well-known exploration-exploitation trade-off: whether to keep trying to acquire new information (explore), or to act consistently with accumulated information to maximize reward (exploit). An efficient solution to this trade-off is essential to obtain acceptable convergence guarantees.

Several different measures of convergence were proposed for the on-line RL problem. These include the number of exploratory episodes [1,6], the number of time steps the agent follows a sub-optimal action, and the number of steps the agent spends following a non-optimal policy [4] (in general, sub-optimality of a policy does not imply sub-optimality of a single action). We shall consider here the latter two, and refer to them as the action-mistake count and policy-mistake count (also known as the *sample complexity of exploration*), respectively. A learning algorithm is said to be PAC (Probable Approximately Correct) if its relevant convergence-rate metric is polynomial in the model size parameters with high probability.

In recent years several PAC algorithms were introduced. These include the R-max algorithm [1], further analyzed in [4], the MBIE algorithm [12] and the Delayed Q-learning algorithm [11]. These algorithms do not use any prior knowledge on the model parameters, but rather estimate empirically either the transition probabilities or directly the Q-function for all state-action pairs. As a result, their convergence-rate metrics are at least proportional to the cardinality of the state and action spaces, which may not be acceptable for large problems. Possible approaches to handle such problems include various approximation schemes, and the use of prior knowledge about the system to enhance learning performance.

An effective use of *structural* knowledge about the system has been demonstrated for factored MDPs in [5]. Here we consider the case where a parameterized model of the system in question is available. The potential problem simplification offered by such models can be demonstrated through a simple queueing example. Consider a single-server queue with buffer and server capacity of 100 customers. Assume that the arrival and service processes are Poisson processes with rate parameters λ and μ, respectively. In this model, all transition probabilities are determined by two parameters only. Therefore, although the cardinality of the state space is 100, it is enough to estimate only two parameters in order to find an optimal policy. This observation turns out to be even more acute in case of a queuing system that contains several such queues, say $N > 1$. While the cardinality of the state space grows exponentially to 100^N, which makes learning on a per-state basis infeasible, the number of parameters associated with arrival/service processes grows linearly to $2N$.

Parameterized control models, in which all model parameters are defined in terms of a smaller parameter vector, have been extensively studied in the context of adaptive control, and in particular stochastic adaptive control [7]. However, the results of this research are focused mostly on asymptotic convergence results, rather than on finite time convergence bounds. Some related recent work may be found in [10].

Our focus in this paper is on parameterized system models with a *finite* parameter space. We further consider the discounted reward problem. We present an efficient RL algorithm for this case, called the Parameter Elimination (PEL) algorithm, and show that its total mistake bound grows linearly (up to logarithmic terms) in the size of parameter space, and independently of the size of the state and action spaces.

Essentially, the PEL algorithm is based on eliminating "unlikely" parameters from the list of plausible parameters, J, using the Sequential Probability Ratio

Test (SPRT) [14]. As for action selection, at every step t an "optimistic" parameter is selected from the set J. This parameter is the one that maximized the (discounted) value function from the current state. The current action is then selected as the optimal one for the optimistic parameter.

The current paper focuses on the case of a *finite* parameter set. While this case is of interest on its own, it may also serve as an intermediate step for treating the continuous parameter case via discretization. A detailed treatment of this approach falls beyond the scope of the present paper.

The rest of the paper is organized as follows. In Section 2 we present the model along with some definitions and notations. Section 3 defines the main performance metrics considered in this paper. In Section 4 we present the PEL algorithm and provide our main performance bounds for this algorithm. Section 5 is devoted to the proof of these results. Section 6 discusses several aspects of the obtained error bound and introduces two illustrative examples. In Section 7 we summarize the results obtained so far and discuss future work. Due to space limitations we omit proofs of most intermediate results, and refer the reader to [2] for the complete details.

2 Model Formulation

An MDP M is specified by a five-tuple $\langle S, A, R, p, \eta \rangle$, where S is a finite state space, A is a finite action space, R is a finite reward set, $p : S \times A \to \Delta(S)$ is the transition kernel and $\eta : S \times A \to \Delta(R)$ is the reward distribution function. Here $\Delta(S)$ denotes the set of probability vectors over the set S, and similarly for $\Delta(R)$. Given that at the time step t the state is $s_t \in S$ and the action is $a_t \in A$, the agent receives a random reward $r_t \in R$ with probability $\eta(r_t|s_t, a_t)$ and moves to state $s_{t+1} \in S$ with probability $p(s_{t+1}|s_t, a_t)$.

The observed history until time t is the sequence $h_t \overset{\triangle}{=} \{s_0, a_0, r_0, ..., s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$. A (deterministic) decision rule is a mapping from history to action, namely $\pi_t : H_t \longrightarrow A$, where $H_t = (S \times A \times R)^t \times S$. A policy $\mathcal{A}$ is a collection of decision rules $\{\pi_t\}_{t=0}^{\infty}$ so that $a_t = \pi_t(h_t)$. Note that a (deterministic) learning algorithm is such a policy. Given an initial state s, the policy $\mathcal{A}$ induces a stochastic process $(s_t, a_t, r_t)_{t=0}^{\infty}$ with probability measure $\mathbb{P}^{\mathcal{A},s}$. The expectation operator corresponding to this measure is denoted by $\mathbb{E}^{\mathcal{A},s}$.

Let $V^{\mathcal{A}}(s) \overset{\triangle}{=} \mathbb{E}^{\mathcal{A},s}\left\{\sum_{t=0}^{\infty} \gamma^t r_t\right\}$ denote the discounted return for policy $\mathcal{A}$ from state s. Here $0 < \gamma < 1$ is the discount factor, which we fix from now on. We refer to $V^{\mathcal{A}}(s)$ as the value function for policy $\mathcal{A}$. A policy $\mathcal{A} = \{\pi_t\}_{t=0}^{\infty}$ is called stationary if $\pi_t = \pi$ for all t, where $\pi : S \to A$ is a function of the current state only. It is well known (e.g., [9]) that there exists a deterministic stationary policy π^* which is optimal in sense that $V^{\pi^*}(s) \geq V^{\mathcal{A}}(s)$ for any state s and any policy $\mathcal{A}$. Denote the corresponding optimal value function as $V^*(\cdot)$. Further define the action-value function (or Q-function) for state-action pair (s, a) as $Q^*(s, a) = \bar{r}(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')$, where $\bar{r}(s, a) \overset{\triangle}{=} \sum_{r \in R} r\eta(r|s, a)$.The following equality, known as Bellman equation, holds for any stationary policy π

and state $s \in S$: $V^{\pi}(s) = \bar{r}(s, \pi(s)) + \gamma \sum_{s' \in S} V^{\pi}(s') p(s'|s, \pi(s))$, while the optimal value function of policy $\pi^*(s)$ satisfies $V^*(s) = \max_a Q^*(s, a) = Q^*(s, \pi^*(s))$. Let R_{max} denote an upper bound on the expected one-step reward, so that $\bar{r}_\theta(s, a) \le R_{max}$ for all $\theta \in \Theta$, $s \in S$ and $a \in A$.

In this paper we assume that the true MDP belongs to a known family $\{M_\theta\}_{\theta \in \Theta}$ of parameterized models, where Θ is a finite parameter set. All models in the given family share the same action, reward and state spaces, while their transition and reward probabilities depend on the parameter $\theta \in \Theta$, i.e., $M_\theta =< S, A, R, p_\theta, \eta_\theta >$. For each MDP M_θ we denote by π^*_θ, V^*_θ and Q_θ an optimal stationary policy, the optimal value function and the Q-function, respectively. In case the optimal policy is not unique, we henceforth fix one (arbitrary) selection. The actual model M thus corresponds to some parameter $\theta_0 \in \Theta$, namely $M = M_{\theta_0}$. We refer to θ_0 as the *true parameter*.

3 Performance Metrics

An effective measure of on-line learning efficiency is the number of time steps the algorithm prescribes sub-optimal action. Recall that an optimal action $a^* = \pi^*(s)$ in state s satisfies $Q^*(s, a^*) = V^*(s)$. Hence the difference $V^*(s) - Q^*(s, a)$ quantifies the effect of taking a single suboptimal action a at state s, and thereafter proceeding optimally. We define the *action mistake count* (AMC) as a total number of ϵ-suboptimal state-action pairs visited by the algorithm during its operation, namely, $AMC(\epsilon) \triangleq \sum_{t=0}^{\infty} \mathbb{I}\{Q^*(s_t, a_t) < V^*(s_t) - \epsilon\}$. Note that for ϵ small enough $AMC(\epsilon) = AMC(0)$ (due to the finiteness of the state and action spaces), so that all non-optimal actions are counted.

A more elaborate performance criterion was suggested by Kakade [4] and originally called the "sample complexity of exploration". This criterion relies on a quantification of "sub-optimality" of a policy rather than a single action. We refer to this criterion as the *policy-mistake count* (PMC) and define it in the following way. Denote by $\mathcal{A}$ the policy of the learning algorithm. Let h_τ be the observed history up to time τ, and denote by $V^{\mathcal{A}}(h_\tau) \triangleq \mathbb{E}^{\mathcal{A}, s_0} \left\{ \sum_{j=\tau}^{\infty} \gamma^{j-\tau} r_j \middle| h_\tau \right\}$ the value of the policy $\mathcal{A}$ starting from time τ. The policy mistake count is defined as follows:

Definition 1. *Let ϵ be a positive number. The time step t in said to be an ϵ-**suboptimal step** if $V^{\mathcal{A}}(h_t) < V^*(s_t) - \epsilon$. Equivalently, we say that the learning agent follows an ϵ-**suboptimal policy** at time t. The **policy-mistake count** (PMC) of a learning algorithm is defined as $PMC(\epsilon) \triangleq \sum\limits_{t=0}^{\infty} \mathbb{I}\left\{V^{\mathcal{A}}(h_t) < V^*(s_t) - \epsilon\right\}$.*

For any $\epsilon > 0$ and learning algorithm AMC is dominated by PMC (see Lemma 1 in [2]), i.e., $AMC(\epsilon) \le PMC(\epsilon)$. It follows that any upper bound on PMC also applies to AMC. For this reason we shall focus in the following on PMC alone. We now define the corresponding notion of a PAC algorithm in the following way:

Definition 2. *A learning algorithm A is called* ***PMC-PAC*** *(or just PAC) if, for any positive ϵ and δ, its policy-mistake count (action-mistake count) is polynomial in $(|\Theta|, \epsilon^{-1}, \delta^{-1}, (1-\gamma)^{-1})$ with probability of at least $(1-\delta)$.*

4 The Parameter Elimination Algorithm

In discrete parameterized models, the learning problem may be reduced to the identification of the true parameter or, at least, a parameter that leads to a near-optimal control policy for the true model. Equivalently, one may try to eliminate all other parameters from the set of optional parameters.

Define the log-likelihood function of the observation $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$ at time step t as

$$l_t(\theta) = \log p_\theta(s_t|s_{t-1}, a_{t-1}) + \log \eta_\theta(r_{t-1}|s_{t-1}, a_{t-1}). \tag{4.1}$$

The *cumulative* log-likelihood is then $G_t(\theta) = \sum_{i=1}^{t} l_t(\theta)$.

Algorithm 1. Parameter ELimination

```
Input: $\{M_\theta\}_{\theta\in\Theta}$ – the finite family of possible MDPs, $\delta$ – an allowed probability of error.
Initialize: Initialize the list of plausible parameter values to $J_0 = \Theta$. Initialize the array of cumulative log-likelihood to $G_0(\theta) = 0$ for all $\theta \in \Theta$.
For $t = 0, 1, ...$ do
1. Stopping condition: If $J_t$ is a singleton, namely $J_t = \{\theta\}$, then use the corresponding policy $\pi^*_\theta$ indefinitely and skip items (2)-(5) below.
2. Find an optimistic parameter: Select a parameter value that maximizes the value function among plausible parameter values: $\theta(t) := \arg\max_{\theta\in J_t} V^*_\theta(s_t)$.
3. Act: Execute the action according to the optimal policy for the optimistic parameter: $a_t := \pi^*_{\theta(t)}(s_t)$.
4. Update: Observe the reward $r_t$ and the next state $s_{t+1}$. Update for all $\theta \in J_t$: $G_{t+1}(\theta) := G_t(\theta) + l_{t+1}(\theta)$ where $l_{t+1}$ is defined in (4.1).
5. Eliminate: Set $J_{t+1} := J_t$ and do:
    a. For all $\theta \in J_{t+1}$ so that $G_{t+1}(\theta) = -\infty$, let $J_{t+1} := J_{t+1} \setminus \{\theta\}$.
    b. Find the most likely parameter in the plausible set $\hat{\theta} := \arg\max_{\theta\in J_{t+1}} G_{t+1}(\theta)$.
    c. For all $\theta \in J_{t+1}$ so that $G_{t+1}(\hat{\theta}) - G_{t+1}(\theta) > \log\left[\frac{3(|\Theta|-1)}{\delta}\right]$, let $J_{t+1} := J_{t+1} \setminus \{\theta\}$.
```

The PEL algorithm proceeds as follows (see Algorithm 1 for details). As an input, the algorithm requires the finite family of possible MDPs $\{M_\theta\}_{\theta\in\Theta}$, with common state, reward and action spaces. The value function $V^*_\theta(\cdot)$ and the optimal policy $\pi^*_\theta(\cdot)$ for each model can be calculated using one of the standard algorithms, i.e., value iteration, policy iteration or linear programming (see [9]). An allowed probability of error δ is also provided as input.

The algorithm maintains a list of plausible parameters J_t throughout its execution. Initially, all parameter values are considered plausible and then they are eliminated one by one. The elimination step is based on the Sequential Probability Ratio Test (SPRT), namely, comparing the log-likelihood ratio $G_t(\theta_i) - G_t(\theta_j)$ to a given threshold $G_{th} > 0$. If at time step t there exist parameters $\theta_i, \theta_j \in J_t$ so that $G(\theta_i) - G(\theta_j) > G_{th}$ then θ_j is eliminated. Equivalently, we first find $\hat{\theta}$, the most likely parameter in the set J_t, and then compare the likelihood of all other plausible parameters to $G(\hat{\theta})$. As the error probability of each elimination can be upper bounded by $e^{-G_{th}}$, the selection of $G_{th} = \log\left[\frac{3(|\Theta|-1)}{\delta}\right]$ yields cumulative error probability of all eliminations less than $\frac{\delta}{3}$ (see section 5.4 for details).

The exploration-exploitation tradeoff is addressed using the so-called "optimism in face of uncertainty" principle. At each time step t, the PEL algorithm selects an "optimistic" action in the following sense. First, the algorithm selects the parameter $\theta(t) \in J_t$ that maximizes the value function $V^*_\theta(s_t)$ for the current state s_t. The selected action is then the optimal one given $\theta(t)$, i.e., $a_t = \pi^*_{\theta(t)}(s_t)$. We note that the selected action may correspond to a different parameter θ at each state, even if the set J_t does not change.

The main result of the paper is the following one.

Theorem 1. *Consider the PEL algorithm with parameter* $0 < \epsilon < \frac{R_{max}}{(1-\gamma)}$ *and* $0 < \delta < 1$. *With probability of at least* $1-\delta$, *PEL's policy-mistake count is upper bounded by*

$$PMC(\epsilon) \leq |\Theta| \frac{R^3_{max}}{\epsilon^3(1-\gamma)^6} L(|\Theta|, \epsilon, \delta, \gamma), \tag{4.2}$$

where $L(|\Theta|, \epsilon, \delta, \gamma) = 801 \log\left(\frac{3|\Theta|}{\delta}\right) \log \frac{4R_{max}}{\epsilon(1-\gamma)}$.

This theorem implies that the PEL algorithm is PAC in terms of the total mistake bound, and its PMC is linear (up to the logarithmic term $L(\cdot)$) in the size of the parameter set. Note that the bound is independent of the cardinality of the state and action spaces.

5 Proof of the Main Result

An outline of the proof of Theorem 1 is as follows. We begin in Section 5.1 by introducing an optimistic auxiliary model that will prove useful later on. In Section 5.2 we define *informative state-action pairs* (Definition 3) that are roughly state-action pairs that distinguish the true MDP and the auxiliary model. We next show in Lemma 1 that within a finite time interval following an ϵ-suboptimal time step (Definition 1), there is a positive probability to reach an informative state-action pair. Moreover, Lemma 4 (Section 5.3) implies that the number of ϵ-suboptimal steps encountered is bounded with high probability in terms of number of actual visits to informative state-action pairs. Hence, once we show that the number of visits to informative state-action pairs is bounded, we can conclude that the policy-mistake count is bounded as well. To show the former, we bound in Section 5.4 the stopping time of the SPRT test (for any fixed parameter $\theta \neq \theta_0$) using a non-decreasing measure of accumulated statistical information related to Bhattacharyya's information coefficient. In Section 5.5 we show that each visit to an informative state-action pair adds some strictly positive amount of information to one parameter at least. Hence the number of visits needed for SPRT to trigger is bounded. Using the pigeon-hole principle, we obtain that the number of visits to an informative state action pairs until convergence to an ϵ-optimal policy is also bounded, thus concluding the proof.

Note that from this point on all the probabilities and expectations refer to the stochastic process induced by the PEL algorithm on the actual MDP M_{θ_0}, unless mentioned otherwise.

5.1 An Auxiliary Model

Consider some *fixed* subset of parameters $J \subseteq \Theta$. For every $s \in S$, define the *optimistic parameter* in J as $\theta(J,s) = \arg\max_{\theta \in J} V_\theta^*(s)$ (with ties decided arbitrarily). Define an auxiliary MDP $M_J = \langle S, A, R, p_J, \eta_J \rangle$, where $p_J(s'|s,a) = p_{\theta(J,s)}(s'|s,a)$ and $\eta_J(r'|s,a) = \eta_{\theta(J,s)}(r'|s,a)$. Further, define the following stationary policy: $\pi_J(s) = \pi^*_{\theta(J,s)}(s)$. This policy picks at each state the optimal action according to the parameter $\theta(J,s)$ that is optimistic for that state. (In the context of the PEL algorithm, it is evident that as long as the set J_t is equal to J, the algorithm follows this stationary policy.) Denote the value function of the MDP M_J under the policy π_J as $V_J^{\pi_J}$. For notational convenience we use the abbreviated notation V_J. Then the auxiliary model is optimistic in the following sense (see [2] for the detailed proof):

Lemma 1. *For any $s \in S$ and $\theta \in J$ it holds that*[1] $V_J(s) \geq V_\theta^*(s)$.

5.2 Implicit Explore or Exploit

We next prove that the PEL algorithm implicitly provides a tradeoff between possible exploration and exploitation. In other words, the agent either follows an ϵ-optimal policy or otherwise gains some information with a positive probability.

The proof is partially based on results from [12] and [6]. For a stationary policy π denote the H-step value function by $V^\pi(s,H) \overset{\triangle}{=} \mathbb{E}^{\pi,s}\left\{\sum_{t=0}^{H-1}\gamma^t r_t\right\}$. The first lemma addresses the sensitivity of the value function to the time horizon.

Lemma 2. *If $H \geq \frac{1}{1-\gamma}\log\frac{R_{max}}{\epsilon(1-\gamma)}$ then $|V^\pi(s,H) - V^\pi(s)| \leq \epsilon$ for all policies π and states s.*

Proof. The result follows easily by bounding the tail of sum of rewards in the definition of the value function (see e.g. Lemma 2 in [6]).

In the following we use $T_{\text{eff}} = \frac{1}{1-\gamma}\log\frac{4R_{max}}{\epsilon(1-\gamma)}$ as an effective horizon length, beyond which the effect on the discounted return is negligible.

[1] Note that the auxiliary model M_J need not be in the family $\{M_\theta\}_{\theta \in \Theta}$. Hence, it may even hold that $V_J(s) > V_\theta^*(s)$ for every $\theta \in \Theta$ and $s \in S$.

The following lemma bounds the sensitivity of the discounted reward function to perturbations in the transition and reward probabilities. For two probability distributions p and q on the finite set A, we use the l_1 norm to measure their separation: $\|p(\cdot) - q(\cdot)\|_1 = \sum_{a \in A} |p(a) - q(a)|$.

Lemma 3. *Let $M_1 =< S, A, R, p_1, \eta_1 >$ and $M_2 =< S, A, R, p_2, \eta_2 >$ be two MDPs with non-negative rewards bounded by R_{max}. Let π be some stationary policy and let ϵ be a positive number. If $\|\eta_1(\cdot|s,a) - \eta_2(\cdot|s,a)\|_1 \le \frac{\epsilon(1-\gamma)^2}{R_{max}}$ and $\|p_1(\cdot|s,a) - p_2(\cdot|s,a)\|_1 \le \frac{\epsilon(1-\gamma)^2}{R_{max}}$ for all states s and actions a, then $\max_{s\in S} \left|V^{\pi}_{M_1}(s) - V^{\pi}_{M_2}(s)\right| \le \epsilon$.*

Proof. Follows from Lemma 4 in [12], after noting that $|\bar{r}_1(s,a) - \bar{r}_2(s,a)| \le R_{max} \|\eta_1(\cdot|s,a) - \eta_2(\cdot|s,a)\|_1$.

To state the central result of this subsection, define informative state-action pairs as those pairs for which either the state transition or the reward distribution are distinct under the true and optimistic models. More precisely:

Definition 3. *Recall that θ_0 is the true parameter. Let $\theta(J,s)$ be defined as in Subsection 5.1. For $t \ge 0$, let K_t be the set of state-action pairs (s,a) for which $\left\|\eta_{\theta(J_t,s)}(\cdot|s,a) - \eta_{\theta_0}(\cdot|s,a)\right\|_1 \le \frac{\epsilon(1-\gamma)^2}{4R_{max}}$, and $\left\|p_{\theta(J_t,s)}(\cdot|s,a) - p_{\theta_0}(\cdot|s,a)\right\|_1 \le \frac{\epsilon(1-\gamma)^2}{4R_{max}}$. We say that the PEL algorithm visited an* ***informative state-action pair*** *at time t, if $(s_t, a_t) \notin K_t$.*

The following proposition asserts that occurrence of an ϵ-suboptimal step leads to an explorative interval, where an informative state-action pair is visited with probability of at least $\frac{\epsilon(1-\gamma)}{2R_{max}}$. Recalling the definition of an ϵ-suboptimal time step in Definition 1, let

$$E_1(t) \triangleq \{\theta_0 \in J_t\} \cap \{V^{\mathcal{A}_t}(h_t) < V^*_{\theta_0}(s_t) - \epsilon\},\ t \ge 0 \tag{5.1}$$

denote the event that the action at time step t is ϵ-suboptimal and the true parameter wasn't eliminated before time t. Let

$$E_2(t) \triangleq \{(s_{t-1}, a_{t-1}) \notin K_{t-1}\} \cup \{J_t \ne J_{t-1}\},\ t \ge 1 \tag{5.2}$$

be the event that at time step $(t-1)$ either an informative state-action pair was visited or some parameter was eliminated from the set J_{t-1} of plausible parameters at time t. Denote by $E_3(t) \triangleq \bigcup_{\tau=t+1}^{t+T_{\text{eff}}} E_2(\tau)$ the event that the informative event $E_2(\tau)$ occurred for τ between $(t+1)$ and $(t+T_{\text{eff}})$. Let $\mathcal{F}_t \triangleq \sigma\{h_t\}$ be the sigma algebra of the history sequence until time step t. Then $E_1(t), E_2(t) \in \mathcal{F}_t$, while $E_3(t) \in \mathcal{F}_{t+T_{\text{eff}}}$.

Proposition 1. *For every t and history h_t that satisfies $E_1(t)$, $\mathbb{P}^{\mathcal{A},s_0}\{E_3(t)|\,h_t\} > \frac{\epsilon(1-\gamma)}{2R_{max}}$.*

Proof. See [2] for the complete proof.

5.3 Discovery Lemma

Proposition 1 shows that in the T_{eff} steps following an ϵ-suboptimal step there is a probability of at least $\frac{\epsilon(1-\gamma)}{2R_{max}}$ to reach some informative state-action pair or eliminate some parameter from J_t. Based on that, Lemma 4 below bounds the number of ϵ-suboptimal steps in terms of the number of actual visits to informative state-action pairs and parameter eliminations.

Let K_t be as in Definition 3 and let N_2 be a positive integer. Recall the definitions of $E_1(t)$, $E_2(t)$, $E_3(t)$ and $\mathcal{F}_t$ from the previous section.

Lemma 4. *For any positive integer N_2, let $T_2(N_2)$ be the time step on which the event $E_2(t)$ occurred for the N_2-th time, namely,*

$$T_2(N_2) = \inf \left\{ n \,\middle|\, \sum_{k=1}^{n} \mathbb{I}\left\{E_2(k)\right\} = N_2 \right\} \tag{5.3}$$

(with $T_2(N_2) = \infty$ is such n does not exist). Then, for all $\epsilon > 0$ and $0 < \delta < 1$, $\mathbb{P}^{\mathcal{A},s_0}\left\{\sum_{k=0}^{T_2(N_2)} \mathbb{I}\left\{E_1(k)\right\} \leq N_1\right\} \geq 1-\delta$, where $N_1 \overset{\triangle}{=} \frac{4R_{max}T_{\text{eff}}}{\epsilon(1-\gamma)}\left[N_2 + \frac{8R_{max}}{\epsilon(1-\gamma)}\log\frac{T_{\text{eff}}}{\delta_3}\right]$.

Proof. See [2] for the complete proof.

5.4 Sequential Hypothesis Testing

The sequential hypothesis test we use in our algorithm was originated by Wald ([14]) and is defined in the following way. Consider a discrete-time stochastic process $\{x_t\}_{t=0}^{\infty}$ taking values in a finite set S. Denote by $x_0^n = \{x_0, ..., x_n\}$ the observations obtained by time n. Let the probability of such observations under hypothesis H_0 be denoted as $p_0(x_0^n)$, and under H_1 as $p_1(x_0^n)$. Note that the discussion here is not limited to Markov processes.

Definition 4. *For any $0 < \delta < 1$ define the stopping time $N^W(\delta) = \inf_n \left\{ n \,\middle|\, \left|\log \frac{p_1(x_0^n)}{p_0(x_0^n)}\right| \geq -\log\delta \right\}$, and the decision rule $d^W(\delta)$ that chooses upon stopping a more likely hypothesis.*

It was shown by Wald ([14]) that the error probability of the SPRT is bounded by δ:

Theorem 2 (Wald). $\mathbb{P}\left\{ d^W(\delta) = H_0 \middle| H_1 \right\} \leq \delta$ *and* $\mathbb{P}\left\{ d^W(\delta) = H_1 \middle| H_0 \right\} \leq \delta$.

We next establish a useful bound on the stopping time of SPRT, using an auxiliary stopping time for the same process based on the Bhattacharyya coefficient rather than the likelihood ratio. We begin by defining the Bhattacharyya coefficient [3].

Definition 5. *Let p and q be probability distributions on the finite set S. Then the **Bhattacharyya coefficient** is $\rho \overset{\triangle}{=} \sum_{s' \in S} p^{1/2}(s')q^{1/2}(s')$.*

Note that $\rho \leq 1$ by the Cauchy-Schwarz inequality. The Bhattacharyya distance (or information) is defined as $-\log\rho$. This metric is related to the l_1-norm of $(p-q)$ in the following way (see [2] for the complete proof):

Lemma 5. $-\log \rho \geq \frac{1}{8} \|p-q\|_1^2$.

Definition 6. *Consider the same processes and hypotheses as in Definition 4. Denote the by* $\rho(x_0^n) = \sum_{x' \in S} p_0^{1/2}(x'|x_0^n) p_1^{1/2}(x'|x_0^n)$ *the Bhattacharyya coefficient between* $p_0(\cdot|x_0^n)$ *and* $p_1(\cdot|x_0^n)$. *Then the* ***Bhattacharyya stopping time*** *(for* $0 < \delta < 1$*) is defined as:* $N^B(\delta) = \inf_n \left\{ n \,\middle|\, \prod_{t=0}^{n-1} \rho(x_0^t) \leq \delta \text{ or } \text{ or } p_1(x_n|x_0^{n-1}) = 0 \right\}$.

We note that the stopping condition $\prod_{t=0}^{n-1} \rho(x_0^t) \leq \delta$ can be written as $R_n \triangleq -\sum_{t=0}^{n-1} \log \rho(x_0^t) \geq -\log \delta$, where R_n is the cumulative Bhattacharyya distance (or total Bhattacharyya information).

While our algorithm uses the Wald test, the Bhattacharyya stopping time will be more handy for analysis as R_n is a non-decreasing sequence. The following proposition relates these two stopping times (see [2] for the complete proof.).

Proposition 2. *For* $0 < \delta < 1$, *the inequality* $\mathbb{P}\left\{ N^W(\delta) > N^B(\delta^{3/2}) \right\} \leq \delta$ *holds both under* H_0 *and* H_1.

5.5 Proof of the Main Result

This subsection builds on our previous results to establish the upper bound on the policy-mistake count (Theorem 1). Consider the PEL algorithm applied to the true MDP M_{θ_0}. The proof proceeds through the following steps. In steps 1-3 we define three "unwanted" events: the event E_4 on which the true parameter θ_0 is eliminated from the plausible parameter set J_t at some point; the event E_5 on which (essentially) there is insufficient number of visits to informative state-action pairs despite a large number of "sub-optimal" steps; and the event E_6 on which a sufficient amount of Bhattacharyya information does not lead to parameter elimination in the SPRT test. We show that the probability of each is bounded by $\frac{\delta}{3}$. In step 4 and step 5 the required upper bound on the PMC is shown to hold on the complement of $E_4 \cup E_5 \cup E_6$. In step 6 we combine the above to conclude the required result.

Step 1: Let $E_4 \triangleq \{\theta_0 \notin \cap_{t=1}^{\infty} J_t\}$ be the event that the actual parameter is eliminated from the set J_t of plausible parameters at some point. As explained in Section 4, the elimination step of the algorithm can be interpreted as a SPRT between any pair of parameter in J_t, with the threshold of $\delta' \triangleq \frac{\delta}{3(|\Theta|-1)}$. From Theorem 2 we obtain that the probability of eliminating θ_0 due to any other fixed parameter is less than δ'. Therefore, by union bound the total probability of eliminating θ_0 is less then $(|\Theta|-1)\delta'$, namely, $\mathbb{P}^{\mathcal{A},s_0}\{E_4\} \leq (|\Theta|-1)\delta' = \frac{\delta}{3}$.

Step 2: Recall the definition of $E_1(t)$ and T_2 from (5.1) and (5.3). Let $E_5 \triangleq \left\{ \sum_{t=1}^{T_2(N_2)} \mathbb{I}\{E_1(t)\} > N_1 \right\}$ be the event that the event $E_1(t)$ was encountered more than N_1 times before the N_2-th occurrence of the event $E_2(t)$. Here, $N_2 \triangleq 12(|\Theta|-1)\left(\frac{4R_{max}}{\epsilon(1-\gamma)^2}\right)^2 \log(\frac{1}{\delta'}) + (|\Theta|-1)$ (this selection is explained in step 4) and N_1 is selected as in Lemma 4 with $\delta := \frac{\delta}{3}$, namely, $N_1 \triangleq \frac{4R_{max}T_{\text{eff}}}{\epsilon(1-\gamma)}\left[N_2 + \frac{8R_{max}}{\epsilon(1-\gamma)} \log \frac{3T_{\text{eff}}}{\delta}\right]$.

Then, Lemma 4 implies (for any N_2 and in particular for the one above), $\mathbb{P}^{\mathcal{A},s_0}\{E_5\} \leq \frac{\delta}{3}$.

Step 3: Consider hypothesis testing between MDPs M_{θ_0} and M_θ for $\theta \neq \theta_0$. Denote by $N^W(\theta,\delta)$, $R_n(\theta)$ and $N^B(\theta,\delta)$ the corresponding SPRT stopping time, the total Bhattacharyya information and the Bhattacharyya stopping time (see Definitions 4 and 6). Let E_6 be the event on which $N^W(\theta,\delta') > N^B\left(\theta,(\delta')^{3/2}\right)$ holds for some $\theta \neq \theta_0$ (i.e., the relation between Bhattacharyya stopping time and SPRT stopping time defined in Lemma 2 is violated). Using Proposition 2 and the union bound we conclude that $\mathbb{P}^{\mathcal{A},s_0}\{E_6\} \leq (|\Theta|-1)\delta' = \frac{\delta}{3}$.

Step 4: Consider a realization $h_\infty = \{s_t,a_t,r_t\}_{t=0}^\infty \in E_4^c \cap E_5^c \cap E_6^c$. Recall the definition of the event $E_2(t)$ in (5.2). We proceed to show that for this realization, $\sum_{t=1}^\infty \mathbb{I}\{E_2(t)\} \leq N_2$.

Let t be a time step on which an informative state-action pair (s_t,a_t) is visited (see Definition 3). Let us assess the Bhattacharyya distance $-\log\rho_t$ between the joint distribution of (r_t,s_{t+1}) under the true model M_{θ_0} and the auxiliary model M_J. Evidently, it equals to the sum of Bhattacharyya distances between $\eta_{\theta_0}(\cdot|s_t,a_t)$ and $\eta_{\theta(t)}(\cdot|s_t,a_t)$, and between $p_{\theta_0}(\cdot|s_t,a_t)$ and $p_{\theta(t)}(\cdot|s_t,a_t)$, where $\theta(t)$ is the optimistic parameter at time t (see Algorithm 1), namely

$$-\log\rho_t = -\log\left[\sum_{s\in S} p_{\theta(t}^{1/2}(s|s_t,a_t)p_{\theta_0}^{1/2}(s|s_t,a_t)\right] - \log\left[\sum_{r\in S}\eta_{\theta(t)}^{1/2}(r|s_t,a_t)\eta_{\theta_0}^{1/2}(r|s_t,a_t)\right].$$

Since $(s_t,a_t) \notin K_t$, it follows by Lemma 5,$-\log\rho_t > \frac{1}{8}\left(\frac{\epsilon(1-\gamma)^2}{4R_{max}}\right)^2 \triangleq B_0$. Hence each visit to an informative state-action pair $(s_t,a_t) \notin K_t$ increases $R_t(\theta)$ by at least B_0 for at least one $\theta \in J_t$. As the sequence $R_t(\theta)$ is non-decreasing, the total number of such increments until the stopping time $N^B(\theta)$ triggers is upper bounded by $\frac{\log((1/\delta')^{3/2})}{B_0}$. By definition, for $h_\infty \notin E_6$ the parameter θ is eliminated no later than $t = N^B\left(\theta,(\delta')^{3/2}\right)$, therefore, by the pigeon-hole principle, the total number of visits to informative state-action pairs until all $\theta \neq \theta_0$ are eliminated from J_t is bounded by $(|\Theta|-1)\frac{\log((1/\delta')^{3/2})}{B_0}$. Recall that $E_2(t)$ occurs if an informative state-action pair was visited at time $(t-1)$ or a parameter was eliminated from J_{t-1}. Hence, $\sum\limits_{t=1}^{\infty}\mathbb{I}\{E_2(t)\} \leq \sum\limits_{t=1}^{\infty}\mathbb{I}\{(s_{t-1},a_{t-1}) \notin K_{t-1}\} + \sum\limits_{t=1}^{\infty}\mathbb{I}\{J_t \neq J_{t-1}\}$ yielding $\sum_{t=1}^\infty \mathbb{I}\{E_2(t)\} \leq (|\Theta|-1)\frac{\log((1/\delta')^{3/2})}{B_0} + (|\Theta|-1) \equiv N_2$.

Step 5: Let T_2, N_2 be as in Step 2. For h_∞ as before we argue that $PMC(\epsilon) \leq N_1$. Since $h_\infty \in E_5^c$, $N_1 \geq \sum\limits_{t=0}^{T_2(N_2)}\mathbb{I}\{E_1(t)\} = \left[\sum\limits_{t=0}^{\infty}\mathbb{I}\{E_1(t)\}\right]\mathbb{I}\{T_2(N_2)=\infty\} + \left[\sum\limits_{t=0}^{\infty}\mathbb{I}\{E_1(t)\}\right]\mathbb{I}\{T_2(N_2)<\infty\} - \left[\sum\limits_{t=T_2(N_2)+1}^{\infty}\mathbb{I}\{E_1(t)\}\right]\mathbb{I}\{T_2(N_2)<\infty\}$. Note that the argument in Step 4 implies, that for $t > T_2(N_2)$ the set J_t of plausible parameters contains only the true parameter θ_0. For this realization the PEL algorithm follows an optimal policy π_{θ_0} from time $T_2(N_2)$ onward, therefore

$\sum_{t=T_2(N_2)+1}^{\infty} \mathbb{I}\{E_1(t)\} = 0$. Hence,$N_1 \geq \sum_{t=0}^{\infty} \mathbb{I}\{E_1(t)\} = \sum_{t=0}^{\infty} \mathbb{I}\left\{V^{\mathcal{A}_t}(h_t) < V^*_{\theta_0}(s_t) - \epsilon\right\}$, where equality holds since $\theta_0 \in J_t$ for realization in E_4^c (see 5.2). Hence, by definition of PMC, $N_1 \geq PMC(\epsilon)$.

Step 6: The bound $N_1 \geq PMC(\epsilon)$ holds on $h_\infty \in E_4^c \cap E_5^c \cap E_6^c$. But, by the union bound, $\mathbb{P}^{\mathcal{A},s_0}\left\{E_4^c \cap E_5^c \cap E_6^c\right\} \geq 1 - \delta$. Substituting N_2 and T_{eff} yields the inequality (4.2) with probability of at least $(1 - \delta)$. □

6 Discussion and Illustrative Examples

In this section we briefly discuss some aspects of the error bound of Theorem 1, and in particular its dependence on the size $|\Theta|$ of the parameter set. As may be seen, the dependence on $|\Theta|$ is essentially linear. The examples below are meant to illustrate two points in this respect: (1) Without further assumptions on the model, linear dependence on $|\Theta|$ is the best that can be obtained by any algorithm. (2) If the model has a favorable representation, the PEL algorithm may have much better error bounds.

Example 6.1. Consider an array of N one-armed bandits $b_1, ..., b_N$, each with a payoff of 0 (loss) or 1 (gain). It is known that exactly one of these bandits has a high gain probability of p_{max}, and all others have a lower gain probability of $p_{min} < p_{max}$. The agent may play any single bandit b_i at each time step. Denote by b^* the optimal bandit with expected payoff p_{max}. Obviously, the optimal policy is to always play this bandit.

Given that the index of the best bandit is initially unknown, our model set contains N different models, namely $|\Theta| = N$. Consider $PMC(\epsilon)$ with ϵ small enough so that a policy mistake occurs each time the agent chooses a suboptimal bandit. In order to minimize the policy mistake count, a learning agent needs to quickly converge to the (initially unknown) bandit b^*. It is easily seen that any learning algorithm may need to try out all N bandits in order to find the best one; thus, the (worst-case) PMC is at least linear in $|\Theta|$ (see [8] for a stronger result).

The next queueing-control example demonstrates that under appropriate conditions, the PEL algorithm makes efficient use of the available statistical information which allows to reduce the dependence of the error bound on the cardinality of Θ. Here this dependence is reduced from linear to logarithmic.

Example 6.2. Consider a discrete time queuing system that consists of N queues $\{Q_1, ..., Q_N\}$, each with a finite buffer size of K and a server S_i. The system is equipped with a router that sends arriving jobs to one of the queues. Let the job arrival process be geometric with a *known* rate λ. Let the service process of each server S_i be geometric with rate $\mu_i \in \mathbb{M}$, where the set $\mathbb{M} \subseteq \mathbb{R}$ of possible service rates is finite and of size M. We assume that the arrival and the service processes are independent of each other and are fully observed.

This system can be modeled as a finite-state finite-action MDP with parametrization vector $(\mu_1, ..., \mu_N) \in \mathbb{M}^N$, hence the PEL algorithm and its analysis apply with $|\Theta| = M^N$. Straightforward substitution in (4.2) yields that $PMC(\epsilon) \leq K_0 M^N$, where $K_0 = \frac{801}{\epsilon^3(1-\gamma)^6} \log \frac{3M^N}{\delta} \log \frac{4}{\epsilon(1-\gamma)}$, with probability of $(1-\delta)$. On the other hand, a refined model-specific analysis produces a tighter bound of $PMC(\epsilon) \leq K_0 N^3$ (with probability $1-\delta$). Thus, the exponential term M^N is replaced with a polynomial term N^3 and the remaining dependence on M is logarithmic.

Due to space limitations we do not provide here the analysis of this bound but rather refer the reader to [2] for the details. The critical observation is that statistical information about the service rate μ_i is obtained whenever the corresponding server S_i is occupied, no matter what the other state components are. Thus, the service rate of the queue is correctly estimated by way of elimination (with high probability) after a certain number of customers have been served in it. On the other hand, it can be shown that a policy-mistake at some stage implies that some customer is to be served in a queue whose rate has not yet been fully estimated. The quantification of these observations yields the bound.

7 Conclusion

Parameterized models offer a great potential for reduction of learning time and cost in large RL problems, alongside less structured methods such as function approximation, aggregation and state abstraction. The former should be used when the available prior information allows to reduce model uncertainty to a lower dimensional parameter space, thereby allowing explicit modeling of inter-state dependencies and avoiding the pitfalls inherent in the local nature of learning in the general, unstructured model.

In this paper we have considered the case of parameterized models with with discrete parameters. We proposed a learning algorithm that incorporates efficient exploration to achieve polynomial mistake bounds in the PAC sense. As may be expected these bounds are independent of the cardinality of the state and action spaces, and in fact may well apply to continuous spaces under reasonable regularity conditions.

Several choices were made in the construction of this algorithm. First, the basic approach taken was that of parameter elimination, rather than on-line parameter estimation. The former has the advantage of reducing the considered parameter set over time, which can quickly converge to a small set if sufficient statistical information is obtained. On the theoretical side, this approach allows the application of sequential hypotheses testing for the analysis of the algorithm. Furthermore, the possible error of eliminating the true parameter cannot be rectified later, and it is therefore important to keep its probability small. The second choice made in the algorithm is to incorporate an optimistic policy which is defined on a per-state basis, rather than freeze a stationary that is optimal for a certain parameter from a certain state. We believe this approach may add to exploration efficiency, although no direct comparison is available. Further work of immediate interest includes the extension of the PEL algorithm to continuous parameter spaces though discretization; the development of other (estimation-based) algorithms that may be appropriate for such spaces; the incorporation of computational constraints; and consideration of other learning criteria such as the total regret for the average reward problem.

References

1. Brafman, R.I., Tennenholtz, M.: R-max - a general polynomial time algorithm for near-optimal reinforcement learning. JMLR 3, 213–231 (2002)
2. Dyagilev, K., Mannor, S., Shimkin, N.: Efficient reinforcement learning in parameterized models. Technical report, Technion (2008), http://www.ee.technion.ac.il/people/shimkin/PREPRINTS/PEL_full.pdf
3. Kailath, T.: The divergence and bhattacharyya distance measures in signal selection. IEEE Transactions of Communication Technology com-15(1), 52–60 (1967)
4. Kakade, S.M.: On the Sample Complexity of Reinforcement Learning. Ph.D thesis, University College London (2003)
5. Kearns, M.J., Koller, D.: Efficient reinforcement learning in factored MDPs. In: IJCAI, pp. 740–747 (1999)
6. Kearns, M.J., Singh, S.P.: Near-optimal reinforcement learning in polynomial time. JMLR 49, 209–232 (2002)
7. Kumar, P.R., Varaiya, P.: Stochastic Systems: Estimation, Identification and Adaptive Control. The MIT Press, Cambridge (1998)
8. Mannor, S., Tsitsiklis, J.N.: The sample complexity of exploration in the multi-armed bandit problem. JMLR 5, 623–648 (2004)
9. Puterman, M.L.: Markov Decision Processes. Discrete Stochastic Programming. Wiley, Chichester (1994)
10. Ryabko, D., Hutter, M.: Asymptotic learnability of reinforcement problems with arbitrary dependence. In: Balcázar, J.L., Long, P.M., Stephan, F. (eds.) ALT 2006. LNCS, vol. 4264, pp. 334–347. Springer, Heidelberg (2006)
11. Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: Proceedings of the ICML 2006 (2006)
12. Strehl, A.L., Littman, M.L.: A theoretical analysis of model-based interval estimation. In: Proceedings of ICML 2005, pp. 857–864 (2005)
13. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (1998)
14. Wald, A.: Sequential Analysis. Wiley, Chichester (1952)

Regularized Fitted Q-Iteration: Application to Planning

Amir massoud Farahmand[1], Mohammad Ghavamzadeh[1], Csaba Szepesvári[1], and Shie Mannor[2]

[1] Department of Computing Science, University of Alberta,
Edmonton, AB T6G 2E8, Canada
{amir,mgh,szepesva}@cs.ualberta.ca
[2] Department of Electrical & Computer Engineering, McGill University,
Montreal, QC H3A 2A7, Canada
shie.mannor@mcgill.ca

Abstract. We consider planning in a Markovian decision problem, i.e., the problem of finding a good policy given access to a generative model of the environment. We propose to use fitted Q-iteration with penalized (or regularized) least-squares regression as the regression subroutine to address the problem of controlling model-complexity. The algorithm is presented in detail for the case when the function space is a reproducing-kernel Hilbert space underlying a user-chosen kernel function. We derive bounds on the quality of the solution and argue that data-dependent penalties can lead to almost optimal performance. A simple example is used to illustrate the benefits of using a penalized procedure.

1 Introduction

We consider planning in a discounted Markovian Decision Problem (MDP) with continuous state space and finite action space. We assume that transitions can be generated at any selected state for any given action. The algorithm that we consider is fitted Q-iteration (e.g., [8]), an instance of sample-based approximate dynamic programming.

The algorithm's main distinguishing characteristic is that the value function iterates are obtained by solving appropriately defined *regularized* least-squares regression problems. We give the particular form of the algorithm when the value functions considered in the iterations belong to some Reproducing Kernel Hilbert Space (RKHS). Our main theoretical results bound the quality of the solutions as a function of the number of samples used by the algorithm, the relation of the RKHS and the MDP and the number of samples used. As usual when regularization is employed, performance is tuned through the choice of a single scalar parameter, the penalty factor, which, in turn, can be selected in a data dependent manner to optimize the performance.

The rationale of studying the use of regularization in solving MDPs is that regularization has proven to be an extremely effective tool in machine learning, in particular in supervised learning. The main idea underlying regularization is to achieve model selection by considering the complexity of solution candidates individually. This is done by adding an appropriate complexity penalty, multiplied by the so-called regularization coefficient, to the empirical risk functional. When the reguralization coefficient is chosen in an appropriate way (based on the data or by complexity regularization), *automatic adaptation* to the complexity of the target function becomes possible: The rate of convergence of such a method is almost as fast as if the complexity of the target function was known beforehand (e.g., Theorem 21.2 of [10]).

As the regularization coefficient effectively controls the size of the function space where the solutions are sought in, our approach of using reguralization in fitted Q-iteration can be considered as a way of tuning the function approximator in an approximate dynamic programming procedure. Recently this tuning problem has received considerable attention (e.g., [7, 8, 13, 15, 18, 19]). However, none of the previous works that we know of explored in a systematic manner how regularization influences the performance of the resulting procedure. The only works that we know of that used regularization are that of Jung and Polani [11], Loth et al. [14] and Xu et al. [22]. In particular, Jung and Polani [11] explored penalizing the empirical L^2-norm of the Bellman-residual for finding the value function of a policy given a trajectory in a *deterministic* system, while L^1-penalties for the same problem were considered by Loth et al. [14]. As the straightforward implementation of penalized least-squares involves a nontrivial computational cost, both papers focused on computational efficiency. Xu et al. [22], on the other hand, used sparsification in Least Squares Temporal Difference learning (LSTD) as an implicit form of regularization and studied the performance of the resulting algorithm experimentally. In our more recent work, we analyzed Regularized Policy Iteration methods that use LSTD and a modified version of Bellman Residual Minimization (BRM)and provided finite time performance bounds [9].

Works where planning in generative models were considered include those of Kearns et al. [12] and Ng and Jordan [17]. Our work is complementary: Our method is guaranteed to achieve optimality in the limit (unlike [17] where policy search with a fixed policy class is considered) and it does not scale exponentially with the effective planning horizon (unlike the lookahead tree building method of [12]). However, our method comes with other restrictions: The MDP has to be sufficiently regular in a sense that will be discussed later. The immediate precursor of this work is that of Munos and Szepesvári [16], where fitted value-iteration was studied in the same framework. In contrast to the approach followed here, Munos and Szepesvári considered state-value functions and they did not study regularization. Although our toolkit is the same, due to these differences our results are different than those in [16], as will be further discussed below.

1.1 The Organization of the Paper

We present the notations and the necessary background on MDPs in Section 2. The fitted Q-iteration algorithm is recalled in Section 3. The first main result that relates the performance of the eventual policy and the L^p-norms of the errors committed during the iterations is presented in Section 4. This result is in turn used in Section 5 to arrive at specific bounds for L^2 regularization. The behavior of the algorithm and the tradeoffs involved are illustrated on a simple domain in Section 6.

2 Background and Notation

Because we consider continuous state spaces, we need a few concepts from analysis. These are introduced first. This is followed by the introduction of the notation and concepts used in connection to MDPs. We refer the reader to Bertsekas and Shreve [4] for further details in connection to these.

For a measurable space with domain S, we let $\mathcal{M}(S)$ denote the set of probability measures over S. For $p \geq 1$, a probability measure $\nu \in \mathcal{M}(S)$, and a measurable function $f : S \to \mathbb{R}$, we let $\|f\|_{p,\nu}$ denote the $L^p(\nu)$-norm of f:

$$\|f\|_{p,\nu}^p = \int |f(s)|^p \nu(ds).$$

For brevity, we shall write $\|f\|_\nu$ to denote the $L^2(\nu)$-norm of f. The supremum-norm, $\|f\|_\infty$, of f is defined by $\|f\|_\infty = \sup_{x \in \mathcal{X}} |f(x)|$. We denote the space of bounded measurable functions with domain $\mathcal{X}$ by $B(\mathcal{X})$, and the space of measurable functions with bound $0 < K < \infty$ by $B(\mathcal{X}; K)$.

A finite-action discounted MDP is defined by a quintuple $(\mathcal{X}, \mathcal{A}, P, S, \gamma)$, where $\mathcal{X}$ is the (possibly infinite) *state space*, $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ is the finite set of *actions*, $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the *transition probability kernel* with $P(\cdot|x,a)$ being the next-state distribution upon taking action a in state x, $S(\cdot|x,a)$ gives the corresponding distribution of *immediate rewards* and $\gamma \in (0,1)$ is the *discount factor*. We make the following assumptions on the MDP:

Assumption A1 . $\mathcal{X}$ is a compact subset of the d-dimensional Euclidean space. We assume that the random immediate rewards are between $-\hat{R}_{\max}$ and $\hat{R}_{\max}$, and the expected immediate rewards $r(x,a) = \int r S(dr|x,a)$ are bounded by $R_{\max}$: $\|r\|_\infty \leq R_{\max}$. (Note that $R_{\max} \leq \hat{R}_{\max}$.)

A *stationary Markov policy* is specified by a measurable mapping $\pi : \mathcal{X} \to \mathcal{M}(\mathcal{A})$. Such a policy and a random initial state $X_0 \in \mathcal{X}$ gives rise to a random trajectory $(X_t, A_t, R_t)_{t \in \mathbb{N}}$ that we call a *trajectory* of π: Here $A_t \sim \pi(\cdot|X_t)$, $R_t \sim S(\cdot|X_t, A_t)$ and $X_{t+1} \sim P(\cdot|X_t, A_t)$. A policy is deterministic if $\pi(\cdot|x)$ concentrates on a single action for all states $x \in \mathcal{X}$. Such a policy will be identified with a mapping $\pi : \mathcal{X} \to \mathcal{A}$ in the obvious way. In the rest of this paper, we use the term policy to refer to stationary Markov policies.

The *value* of a policy π when it is started from a state x is defined as the total expected discounted reward that is incurred while the policy is executed:

$$V^\pi(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| X_0 = x \right], \quad x \in \mathcal{X}.$$

Here (X_t, A_t, R_t) is a random trajectory underlying π (signified by the use of π as the subindex of the expectation operator in the definition of V^π), where X_0 is such that the support of its distribution is the full state space $\mathcal{X}$ (otherwise this distribution can be chosen arbitrarily). Function V^π is also called the *state-value function* of policy π. Closely related to V^π is the *action-value function* of π:

$$Q^\pi(x,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t \middle| X_0 = x, A_0 = a\right], \quad (x,a) \in \mathcal{X} \times \mathcal{A}.$$

Here the distribution of X_0 is restricted as previously, A_0 is such that at time zero all actions are selected with positive probability everywhere ($\mathbb{P}\left(A_0 = a | X_0 = x\right) > 0$, $(x,a) \in \mathcal{X} \times \mathcal{A}$), $R_0 \sim S(\cdot|X_0, A_0)$, $X_1 \sim P(\cdot|X_0, A_0)$ and otherwise $(X_t, A_t, R_t)_{t \geq 1}$ is a trajectory underlying π. It is easy to see that for any π, the functions V^π and Q^π are bounded by $R_{\max}/(1-\gamma)$.

Given an MDP, the goal is to find a policy that attains the best possible values,

$$V^*(x) = \sup_\pi V^\pi(x)$$

simultaneously for all states $x \in \mathcal{X}$. A policy achieving this goal (i.e., $V^\pi = V^*$) is called an *optimal policy.* Function V^* is called the *optimal value function.*

In order to characterize optimal policies let us define the *optimal action-value function*,

$$Q^*(x,a) = \sup_\pi Q^\pi(x,a), \quad (x,a) \in \mathcal{X} \times \mathcal{A},$$

and the concept of *greedy* policies: A deterministic policy π is *greedy* w.r.t. an action-value function $Q \in B(\mathcal{X} \times \mathcal{A})$ if, for all $x \in \mathcal{X}$ and $a \in \mathcal{A}$, $\pi(x) \in \operatorname{argmax}_{a \in \mathcal{A}} Q(x,a)$. Although greedy policies are non-unique, we will write (by slightly abusing the notation) $\pi = \hat{\pi}(.;Q)$. Because $\mathcal{A}$ is finite, a greedy policy always exists no matter how Q is chosen. The importance of Q^* is that any greedy policy w.r.t. Q^* is optimal. Hence, to find an optimal policy it suffices to determine Q^*.

The *Bellman optimality operator* $T : B(\mathcal{X} \times \mathcal{A}) \to B(\mathcal{X} \times \mathcal{A})$ is defined by

$$(TQ)(x,a) = r(x,a) + \gamma \int \max_{a' \in \mathcal{A}} Q(y,a')P(dy|x,a).$$

As it is well known, T is a contraction operator w.r.t. the supremum-norm with index γ: $\|TQ - TQ'\|_\infty \leq \gamma \|Q - Q'\|_\infty$, $Q, Q' \in B(\mathcal{X})$. Moreover, the optimal action-value function is the unique fixed point of T: $TQ^* = Q^*$. Starting from any $Q_0 \in B(\mathcal{X} \times \mathcal{A})$, $Q_{k+1} = TQ_k$ is thus guaranteed to converge (at an exponential rate) to Q^*. This procedure is called *value iteration.*

Throughout the paper we will use $\mathcal{F} \subset \{\, f : \mathcal{X} \to \mathbb{R} \,\}$ to denote some subset of real-valued functions over the state-space $\mathcal{X}$. For convenience, we will treat elements of $\mathcal{F}^M$ as real-valued functions f defined over $\mathcal{X} \times \mathcal{A}$ with the obvious identification $f \equiv (f_1, \ldots, f_M)$, $f(x, a_j) = f_j(x)$, $j = 1, \ldots, M$ (note that M denotes the number of actions). The set $\mathcal{F}^M$ will be the set of admissible functions used in the optimization step of our algorithm.

```
**FittedQ**($D$,$K$,$Q_0$)
// $D$: samples
// $K$: number of iterations
// $Q_0$: Initial action-value function
$Q \leftarrow Q_0$ // Initialization
for $k = 0$ to $K - 1$ do
    $Q' \leftarrow \mathrm{FitQ}(Q, D, k)$
    $Q \leftarrow Q'$
end for
return $Q$
```

Fig. 1. Fitted Q-Iteration

3 Algorithm

The algorithm studied in this paper is an instance of the generic fitted Q-iteration method (e.g., [8]), whose pseudo-code is shown in Fig. 1. The algorithm attempts to approximate the optimal action-value function Q^* and mimics value iteration. Since computing the Bellman operator applied to the last iterate at any point involves evaluating a high-dimensional integral, we use a Monte-Carlo approximation together with a regression procedure. For this purpose a set of samples, D is generated: $D = \{(X_1, A_1, R_1, X'_1), \ldots, (X_N, A_N, R_N, X'_N)\}$. Here, R_t, X'_t are the reward and the next-state when action A_t is chosen in state X_t: $R_t \sim S(\cdot|X_t, A_t)$, $X'_t \sim P(\cdot|X_t, A_t)$. For the sake of simplicity, we assume that the actions are generated by some fixed stochastic stationary policy π_b: $A_t \sim \pi_b(\cdot|X_t)$ and $\{X_t\}$ is an i.i.d. sequence. We will denote the common distribution underlying (X_t, A_t) by ν. The state-marginal of ν is denoted by $\nu_{\mathcal{X}}$. We assume that ν is a strictly positive measure, i.e., its support is $\mathcal{X} \times \mathcal{A}$. Intuitively, this ensures that the samples cover the whole state-action space. In particular for this we must have that $\pi_{b0} \overset{\mathrm{def}}{=} \min_{a\in\mathcal{A}} \inf_{x\in\mathcal{X}} \pi_b(a|x) > 0$.

The fitting procedure FitQ studied in this paper is *penalized least-squares*. Assuming that in the k^{th} iteration we use samples with index $N_k \le i < N_k + M_k = N_{k+1} - 1$, the $(k+1)^{\mathrm{th}}$ iterate is obtained by

$$Q_{k+1} = \operatorname*{argmin}_{Q\in\mathcal{F}^M} \frac{1}{M_k} \sum_{i=N_k}^{N_k+M_k-1} \left[R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(X'_i, a') - Q(X_i, A_i)\right]^2 + \lambda \mathrm{Pen}(Q), \tag{1}$$

where $\mathrm{Pen}(Q)$ is a customary penalty term and $\lambda > 0$ is the regularization coefficient.[1] The first term is the sample-based least-squares error of using $Q(X_i, A_i)$

[1] Note that in practice one would generate the samples whenever they are needed, i.e., there is no need to generate and store all the samples. However, it is also possible to reuse the samples if sample generation is expensive. In such a case the analysis needs to be changed slightly.

to predict $R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(X_i', a')$ at (X_i, A_i). This term is the empirical counterpart to the loss $L_k(Q) = \mathbb{E}\left[(R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(X', a') - Q(X, A))^2\right]$. The minimizer of this loss function is the regression function, which, for any fixed Q_k, in our case is just TQ_k:

$$\mathbb{E}\left[R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(X_i', a') \,|\, X_i = x, A_i = a\right] = (TQ_k)(x, a).$$

As the number of samples grows to infinite, the empirical loss converges to L_k and we expect the iterate Q_{k+1} to converge to TQ_k. To achieve this, one needs to balance the expressiveness of the function class and its complexity (or the resulting function would be overfitted or underfitted). This is the job of the second term on the right hand side of (1). This term implicitly regulates the acceptable complexity of solutions: Choosing larger λ means searching in a smaller space of functions.

When $\mathcal{F}$ is a Sobolev-space[2] of appropriate smoothness order and $\mathrm{Pen}(Q)$ is the corresponding Sobolev-space norm (the L^2-norm of the generalized partials of Q), this optimization leads to thin-plate spline estimates, popular in the non-parametric statistics literature [10]. When searching for a solution in general, the order of smoothness is unknown. Further, the optimal choice of the regularization coefficient would depend on the target function. In order to tune these unknown "parameters" in regression one tries different smoothness orders (this corresponds to choosing the penalty term) with different regularization coefficients and choose the one giving the best empirical error on a hold-out set. The same procedure (though is quite expensive) can be used with fitted Q-iteration. This leads to estimates whose order of rate of convergence is essentially optimal. Further, the convergence rate will scale with the actual roughness, $\mathrm{Pen}(TQ_k)$, of the target function.

Optimizing over a Sobolev-space is a particular case of optimization in a reproducing kernel Hilbert space (RKHS). The latter can be accomplished in a computationally feasible way if one has access to the Mercer kernel function k underlying the RKHS $\mathcal{H}$ and sets $\mathrm{Pen}(Q)$ to be the norm of Q in $\mathcal{H}$ [20]. This way we obtain

$$Q_{k+1} = \operatorname*{argmin}_{Q \in \mathcal{H}} \frac{1}{M_k} \sum_{i=N_k}^{N_k+M_k-1} \left[R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(X_i', a') - Q(X_i, A_i)\right]^2 + \lambda \|Q\|_{\mathcal{H}}^2 . \tag{2}$$

According to the Representer Theorem (see, e.g., Theorem 4.2 in [20]), any solution to Eq. (2) is the sum of kernels centered on the observed samples: i.e.,

$$Q(x, a) = \sum_{i=N_K}^{N_k+M_k-1} \alpha_{i-N_k+1} \mathrm{k}\big((X_i, A_i), (x, a)\big),$$

[2] Sobolev-spaces generalize Hölder spaces, which in turn put constraints on the pointwise smoothness of functions. In particular, Sobolev-spaces allow functions which are only almost everywhere differentiable. Thus, they can be useful for control problems where value-functions often have ridges.

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{M_k})^\top$ are the coefficients that must be determined. Let us assume that Q_k was obtained previously in a similar form:

$$Q_k(x,a) = \sum_{i=N_{k-1}}^{N_{k-1}+M_{k-1}} \alpha^{(k)}_{i-N_{k-1}+1} \mathrm{k}\big((X_i, A_i), (x,a)\big),$$

and let us collect the coefficients in the expansion of Q_k into a vector $\boldsymbol{\alpha}^{(k)} \in \mathbb{R}^{M_{k-1}}$. Replacing Q in Eq. (2) by its expansion and using RKHS properties, we get

$$\boldsymbol{\alpha}^{(k+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^{M_k}} \frac{1}{M_k} \left\| \boldsymbol{r}_k + \gamma \boldsymbol{K}_k^+ \boldsymbol{\alpha}^{(k)} - \boldsymbol{K}_k \boldsymbol{\alpha} \right\|^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K}_k \boldsymbol{\alpha}, \tag{3}$$

with $\boldsymbol{K}_k \in \mathbb{R}^{M_k \times M_k}$, $\boldsymbol{K}_k^+ \in \mathbb{R}^{M_k \times M_{k-1}}$,

$$\begin{aligned}
[\boldsymbol{K}_k]_{ij} &= \mathrm{k}\big((X_{i-1+N_k}, A_{i-1+N_k}), (X_{j-1+N_k}, A_{j-1+N_k})\big), \\
[\boldsymbol{K}_k^+]_{ij} &= \mathrm{k}\big((X'_{i-1+N_k}, A^{(k)}_{i-1+N_k}), (X_{j-1+N_{k-1}}, A_{j-1+N_{k-1}})\big),
\end{aligned}$$

where $A_j^{(k)} = \operatorname{argmax}_{a\in\mathcal{A}} Q_k(X'_j, a)$, and $\boldsymbol{r}_k = (R_{N_k}, \ldots, R_{N_k+M_k-1})^\top$. Solving Eq. (3) for $\boldsymbol{\alpha}$ we obtain

$$\boldsymbol{\alpha}^{(k+1)} = (\boldsymbol{K}_k + M_k \lambda \boldsymbol{I})^{-1} (\boldsymbol{r}_k + \gamma \boldsymbol{K}_k^+ \boldsymbol{\alpha}^{(k)}).$$

The computational complexity of iteration k with a straightforward implementation is $O(M_k^3)$ as it involves the inversion of a matrix. When data is reused between the iterates ($N_k = 1, M_k = N$) only one matrix inversion is necessary as K_k becomes independent of k. However, the total cost as compared when $M_k = N/K$ and when we do K iteration is K^2-times more. Using fast approximate inversion techniques one may get the best of both worlds: Cheaper execution and better use of the samples. In any ways, what remains is to understand how the number of samples influences the quality of the solutions. This is what we study in the next two sections.

4 Error Propagation

In order to analyze how the imperfect fitting procedure influences the final error it is customary to rewrite Fitted Q-iteration in the form

$$\begin{aligned}
Q_{k+1} &= TQ_k - \varepsilon_k, \quad k \ge 0, \\
\varepsilon_{-1} &= Q^* - Q_0.
\end{aligned} \tag{4}$$

Note that these equations define the error sequence ε_k: $\varepsilon_k : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, $\varepsilon_k = TQ_k - Q_{k+1}$. The "initial error function", ε_{-1}, is introduced for the sake of simplifying some expressions that will follow.

The question studied in this section is how the errors $\{\varepsilon_k\}$ influence the performance of the policy greedy w.r.t. Q_K ($K > 0$ is the number of iterations in the algorithm; see Fig. 1). The idea is that the regression procedure controls the size of the error functions ε_k, hence it must be possible to obtain good policies eventually, provided that we can show that if the functions ε_k are "small" in a sense to be specified below then the final error is also small. For $k \geq 0$ let π_k be a greedy policy w.r.t. Q_k: $\pi_k = \hat{\pi}(\cdot; Q_k)$. With this notation our goal is to bound the norm of $V^* - V^{\pi_K} \geq 0$. In order to arrive at such a bound we need the definition of discounted-average concentrability. The motivation for the definition is that we need to relate the norm of errors under ν (the distribution underlying the samples) to the norm ρ chosen by the user (which could be e.g. the uniform distribution).

Definition 1 (Discounted-average Concentrability of Future-State Distributions). *Given $\rho \in \mathcal{M}(\mathcal{X})$, $\nu \in \mathcal{M}(\mathcal{X}\times\mathcal{A})$, $m \geq 0$ and an arbitrary sequence of stationary policies $\{\pi_m\}_{m\geq 1}$ let $\rho^{\pi_1,\ldots,\pi_m} \in \mathcal{M}(\mathcal{X} \times \mathcal{A})$ denote the (future) state-action distribution obtained when the first state is obtained from ρ and then we follow policy π_1, then policy π_2, ..., then π_{m-1} at which step a random action is selected with π_m. Define*

$$c_{\rho,\nu}(m) = \sup_{\pi_1,\ldots,\pi_m} \left\| \frac{d(\rho^{\pi_1,\ldots,\pi_m})}{d\nu} \right\|_\infty ,$$

with the understanding that $c_{\rho,\nu}(m) = \infty$ if the future state-action distribution $\rho^{\pi_1,\ldots,\pi_m}$ is not absolutely continuous w.r.t. ν. The first-order k-shifted ($k \geq 0$, $k \in \mathbb{N}$) discounted-average concentrability of future-state distributions is defined by

$$C^{(1,k)}_{\rho,\nu} = (1-\gamma) \sum_{m=0}^{\infty} \gamma^m c_{\rho,\nu}(m+k).$$

Similarly, the second-order k-shifted ($k \geq 0$, $k \in \mathbb{N}$) discounted-average concentrability of future-state distributions is defined by

$$C^{(2,k)}_{\rho,\nu} = (1-\gamma)^2 \sum_{m=0}^{\infty} m\gamma^{m-1} c_{\rho,\nu}(m+k).$$

In general $c_{\rho,\nu}(m)$ diverges to infinity as $m \to \infty$. However, if the rate of divergence of $c_{\rho,\nu}(m)$ is sub-exponential, i.e., if $\Gamma = \limsup_{m\to\infty} 1/m \log c_{\rho,\nu}(m) \leq 0$ then $C^{(i,j)}_{\rho,\nu}$ will be finite. Note that the definition given here is not identical to the previous similar definition by Munos and Szepesvári [16]. The main difference is that unlike in [16], here ρ is a distribution over the states and ν is a distribution over state-action pairs. The reason is that here we work with action-value functions, while in [16] state-value functions were considered. Note that it is possible to avoid changing this definition (as it was done in Antos et al. 2), but the price is that the bounds will be more conservative. Interestingly, the bounds here avoid the supremum norm arguments used by Antos et al. [2] and are thus less conservative. Note, however, that the arguments presented here do not extend to continuous action spaces studied in [2].

The main result of this section is the following theorem that bounds the loss of using the learned policy π_K as a function of the losses of the solutions of the regression problems solved while running the algorithm:

Theorem 1 (L^p-bound). *Consider a discounted MDP with a finite number of actions. Let $p \geq 1$. Assume that Q_k and ε_k satisfy (4) and that π_k is a policy greedy w.r.t. Q_k. Fix $K > 0$. Define $E_0 = \|\varepsilon_{-1}\|_\infty$ and $\overline{\varepsilon}_K = \max_{0 \leq k \leq K} \|\varepsilon_k\|_{p,\nu}$. Then,*

$$\|V^* - V^{\pi_K}\|_{p,\rho} \leq \frac{2}{(1-\gamma)^2} \left[\gamma^{\frac{K}{p}} E_0 + \left((1-\gamma) \, (C_{\rho,\nu}^{(1,1)})^{\frac{1}{p}} + \gamma \, (C_{\rho,\nu}^{(2,1)})^{\frac{1}{p}} \right) \overline{\varepsilon}_K \right].$$

5 L^2-Bound for Regularized Kernel-Based Regression

In this section we assume that Q_{k+1} is obtained by solving the RKHS regularization problem of Eq. (2). By using Prop. 3 of Zhou [23], the following generalization of Theorem 21.1 of Györfi et al. [10] to arbitrarily RKHS with smooth kernel functions can be obtained. The result is for the case when $\mathcal{X} = [0,1]^d$, but can be generalized to other compact spaces with "regular" boundaries relatively easily.

Theorem 2. *Assume that $\mathcal{X} = [0,1]^d$, $\mathrm{k} \in \mathrm{Lip}^*(s, C(\mathcal{X},\mathcal{X}))$ and Q_k is such that $TQ_k \in \mathcal{H}(=\mathcal{H}_\mathrm{k})$.[3] Furthermore, (for the sake of simplicity) assume that all functions involved in the regression problem (the reward function, Q_k, and the result of the optimization problem Q_{k+1}) are bounded by some constant $L > 0$.[4] Let Q_{k+1} be the solution of (2) with some $\lambda > 0$. Then*

$$\|Q_{k+1} - TQ_k\|_\nu^2 \leq 2\lambda \, \|TQ_k\|_{\mathcal{H}}^2 + \frac{c_1 L^4}{M_k \lambda^{d/s}} + \frac{c_2 \log(1/\delta)}{M_k L^4}$$

holds with probability (w.p.) at least $1-\delta$, for some $c_1, c_2 > 0$.

Note the trade-off in the bound: increasing λ increases the first term, but decreases the second. The optimal choice strikes a balance between these two terms. This choice will depend on the number of samples M_k, the complexity of the target function TQ_k measured by $\|TQ_k\|_{\mathcal{H}}^2$, the dimension d of $\mathcal{X}$, and the degree of smoothness measured by s. With $\lambda = cM_k^{-1/(1+d/s)}$ the rate of convergence is $O(M_k^{-1/(1+d/s)})$, showing that smoother problems give rise to a better rate – an intuitive result. To find the best λ in a data-dependent manner, one may set up a grid of λs, for any given λ generate a new independent sample and choose the λ that gives the lowest risk estimated on the new sample. If the same λ is used in all iterations then a good value can be selected by again setting up a grid and for any value of λ estimate the performance of the obtained policy by following it from a set of start states generated from ρ and then pick the best policy.

[3] For the definition of the generalized Lipschitz space Lip^* see Zhou [23].

[4] When this does not hold, a truncation argument is needed, but the result would essentially be left unchanged.

As an immediate corollary of this result and Theorem 1 we get the following result, assuming that in each iteration we are using the same regularization parameter.

Corollary 3 (L^2-bound). *Assume that the conditions of the previous theorem hold and that we use the same regularization parameter and the same number of samples in each iteration: $M_1 = M_2 = \ldots = M_K$. Let π_K be greedy w.r.t. the K^{th} iterate, Q_K, $B = \max_{0 \le k \le K} \left\| T^k Q_0 \right\|_{\mathcal{H}}^2$. Then, for any $\delta > 0$,*

$$\|V^* - V^{\pi_K}\|_\rho \le \frac{2}{(1-\gamma)^2} \Bigg[\gamma^{\frac{K}{2}} \|\varepsilon_{-1}\|_\infty + \left((1-\gamma)(C_{\rho,\nu}^{(1,1)})^{\frac{1}{2}} + \gamma (C_{\rho,\nu}^{(2,1)})^{\frac{1}{2}}\right) \left[c_1 \lambda B + \frac{c_2 L^4}{M_1 \lambda^{d/s}} + \frac{c_3 \log(K/\delta)}{M_1 L^4} \right]^{1/2} \Bigg]$$

holds w.p. at least $1-\delta$ for some universal constants $c_1, c_2, c_3 > 0$.

Note that by choosing $\lambda = c M_1^{-1/(1+d/s)}$ the second term is made converging to zero with $M_1 \to \infty$ at a rate $O(M_1^{-1/(2(1+d/s))})$, corresponding to the optimal regression rate for smoothness order $s/2$. On the other hand, by choosing larger K, one can make the first term as small as desired.

6 Illustration

In this section, we use a simple illustrative problem that we call the "sinus world" to investigate the behavior of the proposed algorithm. The "sinus world" is designed so as to make it is easy to illustrate the role of the choice of the RKHS, the regularization coefficient, the relation of it to the wiggliness of the reward function or how the noise in the dynamics or that in the observed rewards influences the difficulty of the problem. The state space is $\mathcal{X} = [-5, 5]$ and the problem is to navigate an agent to where rewards are large. The agent can move left or right, its actions are noisy and the boundaries of the state space are absorbing. The discount factor is $\gamma = 0.8$. The reward function is a sine function with some frequency ω. For the details see Table 1.

We used the regularized fitting procedure of Eq. (2) and the kernel function $\mathrm{k}\big((x,a),(x',a')\big) = \mathrm{k}(x,x')\, \mathbb{I}_{\{a=a'\}}$, where the state kernel is Gaussian $\mathrm{k}(x,x') = \exp\big(-\|x-x'\|^2/(2\sigma_{\mathrm{k}}^2)\big)$ with $\sigma_k^2 = 0.1$.[5]

In order to understand what makes the problem difficult for our procedure remember that we use an RKHS norm as the penalty in our fitting procedure. Thus, we expect performance to deteriorate for problems where the target functions, TQ_k, have a larger RKHS norm. With our choice of the kernel, for a function $f : \mathcal{X} \to \mathbb{R}$ we have $\|f\|_{\mathcal{H}}^2 \propto \int |\hat{f}(\omega)|^2 e^{\sigma_{\mathrm{k}}^2 \omega} d\omega$, where $\hat{f}$ is the Fourier transform of f. We see that energies at higher frequencies get exponentially boosted, making the norm to prefer functions with low high frequency content. Now, our target

[5] $\mathbb{I}_{\{E\}}$ denotes the indicator function: $\mathbb{I}_{\{L\}} = 1$ if and only if L is true and $\mathbb{I}_{\{L\}} = 0$, otherwise.

Table 1. The "sinus world". The default parameters are $\omega = 4$, $\sigma_\eta = 0.05$ and $\sigma_r = 1$.

Initial State: $x_0 = -5$

Transitions: $$x_{t+1} = \begin{cases} \hat{x}_{t+1} & \hat{x}_{t+1} \in (-5, +5), \\ -5 & \hat{x}_{t+1} \leq -5, \\ +5 & \hat{x}_{t+1} \geq +5. \end{cases}$$

$$\hat{x}_{t+1} = x_t + a_t + \eta_t \;\; ; \;\; \eta_t \sim \mathcal{N}(0, \sigma_\eta^2), \; ; \;\; a_t \in \{-0.2, 0.2\}$$

Rewards: $r(x_t) = \sin(\omega x_t) + \xi_t,\; \xi_t \sim \mathcal{N}(0, \sigma_r^2)$

functions have the form $TQ = r + \gamma PMQ$, where $Q \in \mathcal{H}$, $M : B(\mathcal{X} \times \mathcal{A}) \to B(\mathcal{X})$ is an operator defined by $(MQ)(x) = \max_{a \in \mathcal{A}} Q(x, a)$, and P is the Markov kernel underlying the dynamics. Operator M generally increases the high frequency content of its input, but operator P (due to the noise of the dynamics) reduces it. Thus, a noisier dynamics helps to reduce $\|TQ\|_{\mathcal{H}}$. However, a noisier dynamics increases the sample variance (i.e., decreases the signal-to-noise ratio), hence we cannot conclude that a noisier dynamics generally helps. On the other hand, the role of the reward function r is largely clear: $\|TQ\|_{\mathcal{H}}$ can be expected to scale with $\|r\|_{\mathcal{H}}$. For our problem, $\hat{r}(\omega)$ is a Dirac function centered at ω. Thus, a larger ω should give rise to more difficult problems. Another source of difficulty is the noise in the observed rewards. This noise does not decrease $\|TQ\|_{\mathcal{H}}$, but only decreases the signal-to-noise ratio.

When solving (2), for the sake of computational efficiency and to increase numerical stability, we used sparsification. In particular, we used the method of Engel et al. [6] which selectively adds state-action pairs to a set of *dictionary* state-action pairs, which are in turn used as a basis for approximating the full solution.[6] The base distribution used to sample the states X_i is uniform. In all cases we used $K = 50$ iterations and the full dataset in all iterations ($N_1 = \ldots = N_K = 1, M_1 = \ldots = M_k = N$).

Performance is evaluated by the relative error, $\max_{a \in \mathcal{A}} \left(\frac{\|Q^*(x,a) - Q_K(x,a)\|_2}{\|Q^*(x,a)\|_\infty} \right)$. Here the L^2 norm is estimated on a grid of 1000 points evenly spaced in the state space. An approximation to the optimal action-value function Q^* is calculated by discretizing the problem using the same regular grid.

Fig. 2(a) (Fig. 2(b)) show the performance of our algorithm as a function of the regularization coefficient λ, for three values of ω (resp., σ_r). All curves are averaged over 30 independent runs of the experiments (across algorithms the same random seeds were used). In Fig. 2(a) the results were obtained using

[6] Sparsification limits the complexity of the fitted functions and consequently acts as implicit regularization, thus reducing the chance of overfitting. This is also apparent from the stability of the curves in the following figures as $\lambda \to 0$.

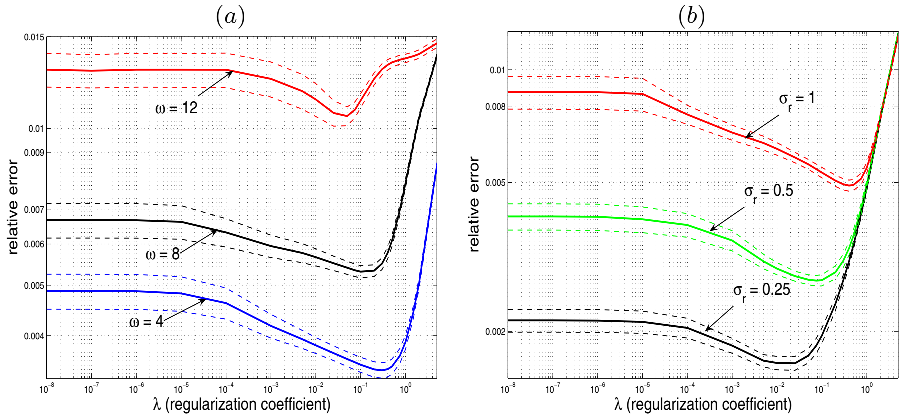


Fig. 2. (a) Effect of changing the reward frequency on the performance of our algorithm. (b) Effect of adding noise to the reward function on the performance of our algorithm.

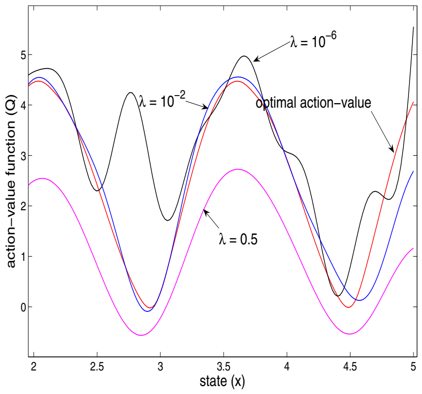


Fig. 3. The optimal action-value function (for the left action) and action-value functions estimated by our algorithm for three values of λ and $N = 200$

$N = 2000$ samples, whereas in Fig. 2(b), the results were obtained using $N = 1000$ samples. On both figures the error bounds are standard error (standard deviation divided by the square-root of the number of runs; here 30).

The results of Fig. 2(a) confirm that increasing the reward frequency increases the generalization error. They also show that for each reward frequency there exists a regularization coefficient λ that attains the minimum error. The minimum is pronounced. Thus, with an appropriate choice of λ (which can be found by e.g. using a hold-out set) significant saving in computation time is possible as one needs less samples to achieve better results. The same conclusion holds for the case when the noise of the observed rewards is varied (Fig. 2(b)).

In order to gain further insight into the behavior of the algorithm we plotted the optimal action-value function for the left action along with the action-value functions found by our algorithm for three different values of λ (Fig. 3). In this experiment, we used $N = 200$ samples. We see that when λ is too small ($\lambda = 10^{-6}$) the procedure overfits, while if λ is too large ($\lambda = 0.5$), underfitting happens. Finally, an intermediate value ($\lambda = 0.01$) gives acceptable fits.

7 Discussion

In this paper we proposed to use penalized least-squares as the regression algorithm in fitted Q-iteration for solving planning problems when a generative model of the environment is available. The problem addressed is that in fitted value iteration and other sample-based planning methods for small sample sizes the function space has to be chosen not only to fit the MDP but also to control over- or underfitting.

As one main contribution of the paper we analyzed the finite-sample performance of the proposed procedure. Although finite-sample performance of fitted value iteration has been considered earlier [1, 2], to the best of our knowledge, this is the first work that addresses finite-sample performance of a *regularized* RL algorithm and gives a concrete algorithm to implement it. The analysis presented here builds on these previous works, but extends and improves them.

As future work, we plan to investigate fitted Q-iteration in multi-kernel situations (different kernel functions correspond to different smoothness classes). Adapting to the situation when the data lies on a low dimensional sub-manifold of the observation space or when certain variables are irrelevant calls for techniques that allow parameterized kernel families. For such a situation ideas of Srebro and Ben-David [21] could be useful. Feature selection could also be addressed by introducing an L^1-penalty in a LASSO-like procedure (e.g., [5]). Another important research topic is to optimize the sample distribution. One idea is to use the estimated action-value function while running the algorithm to actively choose the most informative samples for the next iteration. It would also be desirable to gain experience by applying the proposed method in some realistic problems. Currently, one main limitation is that the procedure is quite expensive to run.

Finally, let us note that even though the results of this paper are presented for planning, the extension to the learning scenario when a good policy is to be learned given a long, representative trajectory of some behavior policy looks possible along the lines of the works [1, 2, 3].

References

[1] Antos, A., Szepesvári, C., Munos, R.: Value-iteration based fitted policy iteration: learning with a single trajectory. In: IEEE ADPRL, pp. 330–337 (2007)

[2] Antos, A., Munos, R., Szepesvári, C.: Fitted Q-iteration in continuous action-space MDPs. In: Advances in Neural Information Processing Systems 20, NIPS 2007 (in print, 2008)

[3] Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. Machine Learning 71, 89–129 (2008)

[4] Bertsekas, D.P., Shreve, S.E.: Stochastic Optimal Control (The Discrete Time Case). Academic Press, New York (1978)

[5] Bunea, F., Tsybakov, A., Wegkamp, M.: Sparsity oracle inequalities for the lasso. Electronic Journal of Statistics 1, 169–194 (2007)

[6] Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. IEEE Transaction on Signal Processing 52(8), 2275–2285 (2004)

[7] Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with Gaussian processes. In: ICML 2005: Proceedings of the 22nd international conference on Machine learning, pp. 201–208. ACM, New York (2005)

[8] Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556 (2005)

[9] Farahmand, A.M., Ghavamzadeh, M., Szepesvári, C., Mannor, S.: Regularized policy iteration. In: Advances in Neural Information Processing Systems 21, NIPS 2008 (to appear, 2008)

[10] Györfi, L., Kohler, M., Krzyżak, A., Walk, H.: A distribution-free theory of nonparametric regression. Springer, New York (2002)

[11] Jung, T., Polani, D.: Least squares SVM for least squares TD learning. In: ECAI, pp. 499–503 (2006)

[12] Kearns, M., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markovian decision processes. In: Proceedings of IJCAI 1999, pp. 1324–1331 (1999)

[13] Lagoudakis, M.G., Parr, R.: Reinforcement learning as classification: Leveraging modern classifiers. In: ICML 2003, pp. 424–431 (2003)

[14] Loth, M., Davy, M., Preux, P.: Sparse temporal difference learning using LASSO. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (2007)

[15] Mannor, S., Menache, I., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research 134, 215–238 (2005)

[16] Munos, R., Szepesvári, C.: Finite-time bounds for fitted value iteration. Journal of Machine Learning Research 9, 815–857 (2008)

[17] Ng, A.Y., Jordan, M.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, pp. 406–415 (2000)

[18] Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning 49, 161–178 (2002)

[19] Parr, R., Painter-Wakefield, C., Li, L., Littman, M.L.: Analyzing feature generation for value-function approximation. In: ICML, pp. 737–744 (2007)

[20] Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002)

[21] Srebro, N., Ben-David, S.: Learning bounds for support vector machines with learned kernels. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS, vol. 4005, pp. 169–183. Springer, Heidelberg (2006)

[22] Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Trans. on Neural Networks 18, 973–992 (2007)

[23] Zhou, D.-X.: Capacity of reproducing kernel spaces in learning theory. IEEE Transactions on Information Theory 49, 1743–1752 (2003)

A Near Optimal Policy for Channel Allocation in Cognitive Radio

Sarah Filippi[1], Olivier Cappé[1], Fabrice Clérot[2], and Eric Moulines[1]

[1] LTCI, TELECOM ParisTech and CNRS, 46 rue Barrault, 75013 Paris, France
{filippi,cappe,moulines}@telecom-paristech.fr
[2] France Telecom R&D, 2 avenue Pierre Marzin, 22300 Lannion, France
fabrice.clerot@orange-ftgroup.com

Abstract. Several tasks of interest in digital communications can be cast into the framework of planning in Partially Observable Markov Decision Processes (POMDP). In this contribution, we consider a previously proposed model for a channel allocation task and develop an approach to compute a near optimal policy. The proposed method is based on approximate (point based) value iteration in a continuous state Markov Decision Process (MDP) which uses a specific internal state as well as an original discretization scheme for the internal points. The obtained results provide interesting insights into the behavior of the optimal policy in the channel allocation model.

1 Introduction

Partially Observable Markov Decision Processes (POMDP) have been widely used for planning problems with uncertainty about the current state. POMDP generalize the standard Markov Decision Processes (MDP) in allowing for a partial observation of the state either due to the presence of observation noise or to the fact that only a part of the state vector is actually observed (censoring) [1,2,3,4]. Such diverse causes of the partial observation lead to many different solutions tailored to the problem at hand: although the general solution is known to be amenable to a continuous state MDP using the belief state formalism [5,6,7], this usually does not lead to tractable solutions.

In the following, we develop and study the performance of a POMDP approach to channel allocation in a cognitive radio system where censoring occurs due to the constraints that only some channels can be sensed at a given time. We compare the performance of the proposed policy with that of the sub-optimal approach introduced by [8] for this model.

The outline of the paper is as follows: Sect. 2 describes the channel allocation problem and casts it into the POMDP formalism; Sect. 3 describes our approach towards a near-optimal policy; Sect. 4 presents our experimental results.

2 Modeling the Channel Allocation Problem for Cognitive Radio

2.1 Channel Allocation

We briefly sketch below the description of this problem as given in [8]: consider a network consisting of N independent channels, with bandwidths B_i, for $i = 1, \dots N$. These N channels are licensed to a primary network whose users communicate according to a synchronous slot structure. At each time slot, channels are either free or occupied (see Fig. 1). The traffic statistics of the primary network are supposed to be known and modeled as a discrete-time Markov chain with known transition probabilities.

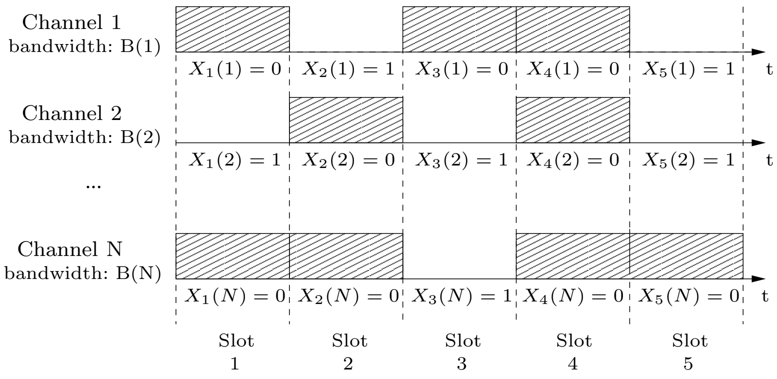


Fig. 1. Representation of the primary network

Consider now a secondary user seeking opportunities of transmitting in the free slots of these N channels without disturbing the primary network, that is to say without transmitting informations in occupied channels. Moreover, this secondary user has not full knowledge of the availability of each channel. In each slot, the secondary user chooses a set of L_1 channels to sense and transmits in L_2 channels among the L_1 observed channels. The aim of the secondary user is to leverage this partial observation of the channels so as to maximize its throughput.

2.2 POMDP Modeling

This problem can be modeled by a POMDP. At the beginning of the slot t, the network state is $[X_t(1), \dots, X_t(N)]'$, where $X_t(i)$ is equal to 0 when the channel i is occupied and 1 when the channel is idle. The states of different channels are assumed to be independent, i.e. for $i \neq j$, $X_t(i)$ and $X_t(j)$ are independent. Then, the secondary user selects a set of L_1 channels to sense. This choice corresponds to an action $A_t = [A_t(1), \dots, A_t(N)]'$, where the component $A_t(i) = 1$ if the i-th channel is sensed and $A_t(i) = 0$ otherwise. Since L_1 channels are observed at each time slot, $\sum_{i=1}^{N} A_t(i) = L_1$. The observation is an N-dimensional vector $[Y_t(1), \dots, Y_t(N)]'$ such that, for the i-th channel:

$$Y_t(i) = \begin{cases} X_t(i) & \text{if } A_t(i) = 1 \\ \star & \text{otherwise} \end{cases} \tag{1}$$

where $\star$ is an arbitrary number not in $\{0, 1\}$. Based on this observation, the user chooses L_2 channels to access among the L_1 observed channels. He then receives a reward R_t which depends on the action A_t, the observation Y_t and the choice of the channels to access.

At the beginning of the next time slot, the network state is X_{t+1} which only depends on the network state X_t at the beginning of the slot t. Note that the state transition probability does not depend on the actions. Actually, since the secondary user does not disturb the primary network, actions do not affect the state transitions. Assume that the marginal distribution of the state transitions is known. Specifically, channel i transits from state 0 (unavailable) to state 1 (available) with probability $\alpha(i)$ and stays in state 1 with probability $\beta(i)$. In the following, we consider these distributions as being time-homogeneous, potentially restricting our study to a time period where this assumption holds true.

The reward gained at each time slot is equal to the aggregated bandwidth available. Therefore, after observing L_1 channels, the optimal choice for the secondary user is to send through up to L_2 idle observed channels chosen by decreasing bandwidth order and the reward is the sum of those bandwidths. Including (possibly bandwidth-dependent) collision penalties to the reward allows to develop a similar approach if the secondary user must choose exactly L_2 channels to transmit among the L_1 observed channels, hence possibly transmitting through busy channels if there are less than L_2 idle observed channels. If we choose to transmit in every observed channels (i.e. $L_2 = L_1$), the reward is exactly equal to

$$R_t = R(X_t, A_t) = \sum_{i=1}^{N} B(i) \mathbb{1}_{\{A_t(i)=1, X_t(i)=1\}} - c \mathbb{1}_{\{A_t(i)=1, X_t(i)=0\}} ,$$

where c is the collision penalty. The problem therefore reduces to the choice of the L_1 channels to be observed and the secondary user seeks a policy π to maximize the discounted expected reward:

$$\mathrm{E}^{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} R(X_t, A_t) \right] , \quad 0 < \gamma < 1 .$$

2.3 Link with the Restless Multi-armed Bandit Framework

This model may be seen as an instance of the notoriously difficult restless multi-armed bandit problem [9]. The multi-armed bandit problem (MAB) is one of the most fundamental problem in stochastic decision theory. Consider a bandit with N independent arms. For simplicity, assume that each arm may be in one of two states $\{0, 1\}$. At any time step, the player can play L_1 arms. If arm i in state $X_t(i)$ is played, it transitions in a Markovian fashion to state $X_{t+1}(i)$ and yields a reward $R_t(i)$. Contrary to the stochastic MAB problem in which the states of the arms which are not played stay the same, in the restless bandit problem, the states of all the arms vary in a Markovian fashion. The channel allocation problem may be seen as a restless bandit problem, since the states of the channel evolve according to the dynamic imposed by the primary network whether the channel is sensed or not.

In [10], the computational complexity of the problem has been studied: the authors established that the planning task in restless bandit model is PSPACE-hard. Nevertheless, some recent research has put forward sub-optimal index strategies [11,12]. An index strategy consists in separating the optimization task into N channel-specific problems following the idea originaly proposed by Whittle [9].

3 Near Optimal Policy Formulation

3.1 Internal State Definition

In a POMDP, the state of the underlying Markov process is unknown - here the secondary user only observes a limited number of channels at any given time. To choose which channels to observe, the secondary user has to construct an internal state that summarizes all past decisions and observations. A standard approach to solving a POMDP is to introduce the state probability distribution (belief state), which is a 2^N-dimensional vector $\Lambda_t = [\Lambda_t(1), \ldots, \Lambda_t(2^N)]'$ such that

$$\Lambda_t(x) = \mathbb{P}\left[X_t = x \,|\, A_{1:t}, Y_{1:t}, \Lambda_0\right] \ , \quad x \in \mathsf{X} \overset{\text{def}}{=} \{0,1\}^{\times N} \ .$$

We define Λ_0 as the initial state probability. It has been shown that the belief vector is a sufficient internal state [5] i.e.

- there exists a deterministic function such as $\Lambda_{t+1} = \tau(\Lambda_t, Y_{t+1}, A_{t+1})$,
- for every function $f \geq 0$, $\mathrm{E}\left[f(X_t) \,|\, A_{1:t}, Y_{1:t}, \Lambda_0\right] = \mathrm{E}\left[f(X_t) \,|\, \Lambda_t\right]$.

In channel allocation, the independence between the channels can be exploited to construct a N-dimensional sufficient internal state. Let $I_t = [I_t(1), \ldots, I_t(N)]'$ where $I_t(i)$ is the probability, conditioned on the sensing and decision history, that channel i is available at the beginning of slot t:

$$I_t(i) = \mathbb{P}\left[X_t(i) = 1 \,|\, A_{1:t}, Y_{1:t}, I_0\right] \ , \quad i \in \{1, \ldots, N\} \ , \tag{2}$$

and $I_0(i) = \mathbb{P}(X_0(i) = 1)$.

Proposition 1. *If the initial probability Λ_0 can be written as a product of the marginal probabilities i.e. $\forall x \in \mathsf{X}$, $\Lambda_0(x) = \prod_{i=1}^N I_0(i)^{x(i)}(1 - I_0(i))^{1-x(i)}$, then, for all $x \in \mathsf{X}$,*

$$\Lambda_t(x) = \prod_{i=1}^{N} I_t(i)^{x(i)}(1 - I_t(i))^{1-x(i)} \ , \tag{3}$$

and there exists a function $\tau : \mathcal{I} \times \mathsf{A} \times \mathsf{Y} \to \mathcal{I}$ such that, for each component i,

$$\begin{aligned} I_{t+1}(i) &= \tau(I_t, a_{t+1}, y_{t+1})(i) \\ &= \left[\mathbb{1}_{y_{t+1}(i)=1}\right]^{a_{t+1}(i)} \left[I_t(i)\beta(i) + (1 - I_t(i))\alpha(i)\right]^{1-a_{t+1}(i)} . \end{aligned} \tag{4}$$

This proposition shows that the internal state defined in (2) is a sufficient internal state.

3.2 Value Function and Bellman Equation

It is well known that the optimal policy for the POMDP can be derived from the optimal policy, i.e. the choice of actions maximizing the discounted expected reward, in an equivalent MDP where the sufficient internal state of the POMDP plays the role of the state variable.

Let V^π the value function of the policy π:

$$V^\pi(I) = \mathrm{E}^\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \,\middle|\, I_0 = I\right] ,$$

where γ is the discount factor. There exists a deterministic stationary policy $\pi^* = (\pi^*_t)_t$ which is optimal [13]. At each time t, the decision rule $\pi^*_t = \pi^*_0$ is a function from the internal state space $\mathcal{I}$ to the action space A such that

$$V^{\pi^*} = V^* \overset{\text{def}}{=} \max_{\pi \in \Pi} V^\pi ,$$

where Π is the set of the deterministic stationary policy. The optimal value function V^* satisfies the Bellman equation:

$$V^*(I) = \max_{a \in \mathsf{A}} \left\{ \rho(I, a) + \gamma \sum_{y \in \mathsf{Y}} M(I, a; y) V^*(\tau(I, a, y)) \right\} ,$$

where $\rho(I, a)$ is the expected one-step reward given that the internal state is I and the action is a, and $M(I, a; y)$ is the observation probability conditioned by the action and the internal state. More explicitly, for internal state $I_t = I$ and action $A_{t+1} = a$,

$$\rho(I,a) = \sum_{i, a(i)=1} \{(I(i)\beta(i) + (1-I(i))\alpha(i)\}B_i - \{I(i)(1-\beta(i)) + (1-I(i))(1-\alpha(i))\}c ,$$

and $M(I, a; y) = \prod_{i=1}^{N} M_i(I, a; y)$ where

$$M_i(I, a; y) = \begin{cases} I(i)\beta(i) + (1 - I(i))\alpha(i) & \text{if } a(i) = 1 \text{ and } y(i) = 1 \\ I(i)(1 - \beta(i)) + (1 - I(i))(1 - \alpha(i)) & \text{if } a(i) = 1 \text{ and } y(i) = 0 \\ 1 & \text{if } a(i) = 0 \text{ and } y(i) = \star \\ 0 & \text{otherwise} \end{cases} .$$

Note that $M_i(I, a; y)$ depends only on the i-th component of I, a, y, α and β. An optimal policy, named the greedy policy, can be deduced from the optimal value function V^* by

$$\forall I \in \mathcal{I}\ ,\ \pi_0^*(I) = \operatorname*{argmax}_{a \in \mathsf{A}} \left\{ \rho(I, a) + \gamma \sum_{y \in \mathsf{Y}} M(I, a; y) V^*(\tau(I, a, y)) \right\} .$$

The value iteration algorithm is generally used to compute V^*: given an initial value function V_0, we compute iteratively V_n as follows

$$V_n(I) = \max_{a \in \mathsf{A}} \left\{ \rho(I, a) + \gamma \sum_{y \in \mathsf{Y}} M(I, a; y) V_{n-1}(\tau(I, a, y)) \right\} , \quad \forall n \geq 1 .$$

The optimal value function is then obtained as the unique fixed point of this iteration. However, this algorithm is mostly of theoretical interest because the set of internal states is infinite.

3.3 Discretizing the Internal State Space

A pratical solution is to restrict the set of internal points at which we will compute the value function. Truncating the set of possible internal states clearly induces some loss of efficiency, but, as shown below, it however allows to compute a near-optimal policy. There exist many ways to choose the set of internal points but we focus on a solution that takes into account the particular structure of the internal state.

From eq. (4) it follows that for each observed channel i (i.e. such that $A_t(i) = 1$) the i-th component of the intenal state $I_t(i)$ is either equal to 0 or 1 according to the state of the channel. Since L_1 channels are observed at each slot, L_1 components of the internal state are either 0 or 1. In addition, if channel i is not observed, the i-th component of the internal state depends on the last oberved state of the channel. More precisely, for each channel i, $I_t(i) \in \{0, 1, \nu(i), I_0(i), p_k^0(i), p_k^1(i), k > 0\}$, where $\nu(i)$ is the stationary probability that the channel i is idle (in the following, we assume that $I_0 = \nu$) and denote $p_k^0(i)$ (respectively $p_k^1(i)$) the probability that the channel i is idle given that we haven't observed it for k steps and that the last observed state was 0 (respectively 1). The probabilities $p_k^0(i)$ and $p_k^1(i)$ satisfy the following recursions:

$$\begin{aligned} p_k^0(i) =& \mathbb{P}\left[X_t(i) = 1 \,|\, X_{t-k}(i) = 0, A_{t-k+1}(i) = \cdots = A_t(i) = 0\right] \\ =& \begin{cases} \alpha(i) & \text{if } k = 1 \\ p_{k-1}^0(i)\beta(i) + (1 - p_{k-1}^0(i))\alpha(i) & \text{otherwise} \end{cases} , \\ p_k^1(i) =& \mathbb{P}\left[X_t(i) = 1 \,|\, X_{t-k}(i) = 1, A_{t-k+1}(i) = \cdots = A_t(i) = 0\right] \\ =& \begin{cases} \beta(i) & \text{if } k = 1 \\ p_{k-1}^1(i)\beta(i) + (1 - p_{k-1}^1(i))\alpha(i) & \text{otherwise} \end{cases} . \end{aligned}$$

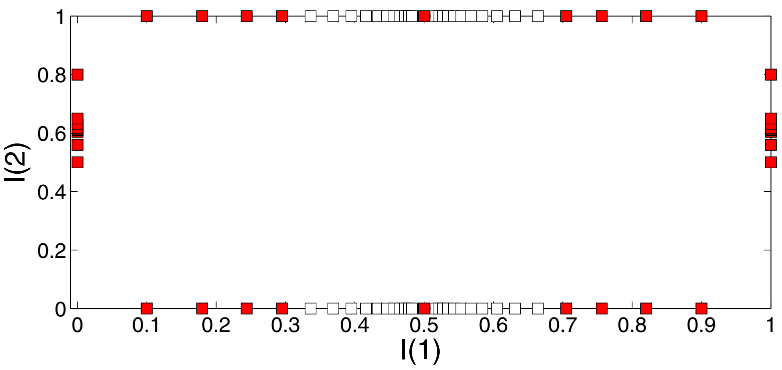


Fig. 2. Representation of all the internal states reachable from the initial internal state I_0 (all the squares) and the set $\tilde{\mathcal{I}}_{I_0,K}$ (the grey squares) for a two-channel model and one channel observed once. We used $\alpha = (0.1, 0.8)'$, $\beta = (0.9, 0.5)'$, and $K = 4$. $I(1)$ and $I(2)$ are respectively the first and the second components of the internal state I.

The set of the internal states $\tilde{\mathcal{I}}_{I_0,K}$ that we construct is composed of all internal points reachable in K steps or less from the initial internal state I_0. We add to this set the stationary probabilities. These internal points are N-dimensional vectors such that $I_t(i) \in \{0, 1, \nu(i), p_k^0(i), p_k^1(i), 0 < k \leq K\}$ subject to the constraint that I_t has exactly L_1 components equals to 0 or 1. The constant K is set so that the size of $\tilde{\mathcal{I}}_{I_0,K}$ remains small enough to keep the computational requirements under a pre-defined level. The set $\tilde{\mathcal{I}}_{I_0,K}$ is displayed in Fig. 2 for a two-channel model and $L_1 = 1$. Note that, in this case, all the elements of $\tilde{\mathcal{I}}_{I_0,K}$ have a component equal to 0 or 1 and that, for a fixed component, the internal points cluster around the stationary probability. The set $\tilde{\mathcal{I}}_{I_0,K}$ may be seen as a kind of adapted grid.

3.4 Algorithm

We can now compute the optimal value function based on the set of internal points $\tilde{\mathcal{I}}_{I_0,K}$ using the value iteration algorithm. Given an initial value function V_0, we compute iteratively V_n as follows

$$\forall I \in \tilde{\mathcal{I}}_{I_0,K} \ , \ \tilde{V}_{n+1}(I) = \max_{a \in \mathsf{A}} \left\{ \rho(I,a) + \gamma \sum_{y \in \mathsf{Y}} M(I,a;y) \tilde{V}_n(\tilde{\tau}(I,a,y)) \right\} \ ,$$

where $\tilde{\tau} : \tilde{\mathcal{I}}_{I_0,K} \times \mathsf{A} \times \mathsf{Y} \to \tilde{\mathcal{I}}_{I_0,K}$ is such that

$$\tilde{\tau}(I,a,y) = \eta(\tau(I,a,y)) = \begin{cases} \tau(I,a,y) & \text{if } \tau(I,a,y) \in \tilde{\mathcal{I}}_{I_0,K} \\ \operatorname{argmin}_{I' \in \mathsf{I}} d(I', \tau(I,a,y)) & \text{otherwise} \end{cases} \ ,$$

where d is the distance defined by

$$d(I, I') = \begin{cases} \sum_{j=1}^{N} |I(j) - I'(j)| & \text{if } \forall i \text{ s.t. } I(i) \in \{0,1\},\, I(i) = I'(i). \\ \infty & \text{otherwise} \end{cases} .$$

The near optimal policy we propose consists in choosing the greedy action with respect to the near optimal value function $\tilde{V}^* = \lim_{n\to\infty} \tilde{V}_n$. The method is summarized algorithm 1. The function *Ipoints* computes a set of internal states as explained in the former section.

Algorithm 1. Algorithm to compute a near optimal policy

```
Require: $N$, $L_1$, $\alpha$, $\beta$, $V_0$, $\epsilon$, $\gamma$
1: $\tilde{\mathcal{I}}_{I_0,K} = Ipoints(N, L_1, \alpha, \beta, K)$
2: while $\|\tilde{V}_n - \tilde{V}_{n+1}\|_2 > \epsilon(1-\gamma)/(2\gamma)$ do
3:     for each $I \in \tilde{\mathcal{I}}_{I_0,K}$ do
           $\tilde{V}_n(I) = \max_{a\in A}\{\rho(I,a) + \gamma \sum_{y\in Y} M(I,a;y)\tilde{V}_{n-1}(\tilde{\tau}(I,a,y))\}$
4:     end for
5:     n=n+1;
6: end while
7: for each $I \in \tilde{\mathcal{I}}_{I_0,K}$ do
       $\pi_0^*(I) = \operatorname{argmax}_{a\in A}\left\{\rho(I_{t-1},a) + \gamma \sum_{y\in Y} M(I_{t-1},a;Y_t)\tilde{V}^*(\tilde{\tau}(I_{t-1},a,Y_t))\right\}$
8: end for
```

3.5 Approximation Result

For this discretization scheme, it is possible to show the following theorem. The proof is omitted for sake of brevity.

Theorem 1. *For all $I \in \tilde{\mathcal{I}}_{I_0,K}$, the error between the optimal value function V^* and the n-th approximate value function $\tilde{V}_n$ is bounded:*

$$\left|V^*(I) - \tilde{V}_n(I)\right| \le C_1 \sum_{i, I(i)\notin\{0,1\}} \frac{|\beta(i) - \alpha(i)|^K}{1 - \beta(i) + \alpha(i)} \max\{\alpha(i), 1-\beta(i)\} + C_2 \frac{\gamma^n}{1-\gamma} ,$$

where C_1 and C_2 are the following constants

$$C_1 = \frac{\max_i |\beta(i) - \alpha(i)|}{1 - \gamma \max_i |\beta(i) - \alpha(i)|} \left(\max_i |B_i + c| + 2\frac{\gamma}{1-\gamma} \sup_{I\in\mathcal{I}, a\in A} |\rho(I,a)| \right) ,$$

$$C_2 = \sup_{I\in\mathcal{I}, a\in A} |\rho(I,a)| .$$

The bound decreases when the discretization parameter K increases, at a faster rate than the bound obtained for generic POMDP [14] who do not exhibit the special structure presented in Sec. 3.1. Note that this decreasing rate depends on $|\beta(i) - \alpha(i)|$ for $i = 1, \ldots, N$. The practical impact of K will be discuss further.

4 Simulation Results

In this section, we present some experimental results. Particularly, we compare the performance of the proposed policy with that of the sub-optimal approach introduced by [8], which consists in choosing the action that maximizes the expected one-step reward:

$$A_t(I) = \underset{a}{\operatorname{argmax}}\, \rho(I, a) = \underset{a}{\operatorname{argmax}}\, \mathrm{E}\left[R_t \mid A_t = a, I_{t-1} = I \right] \ .$$

We will refer to this choice of action as the 1STEP-strategy. In addition, the strategy we propose will be noted NOPT-strategy. Except when noted otherwise, we used $\gamma = 0.9$ and $K = 10$.

For clarity, the results are presented in the two-channel case, assuming that these channels have the same bandwidth $B_i = 1$, for $i = 1, 2$. At each time slot, the secondary user observes one channel and accesses to it if it is idle. We will use different values of the transition probabilities α and β. We are specially interested in networks where some channels are persistent or very fluctuating since the NOPT-strategy will be able to take advantage of it. In the scenario 1, the state of the first channel remains unchanged with a large probability, i.e. $1 - \alpha(1) = \beta(1) = 0.9$, and the probability that the second channel is idle is the same if the channel was idle or occupied in the previous slot. We use $\alpha(2) = \beta(2) = 0.51$. In Fig. 3, we display for both of the studied strategies, the

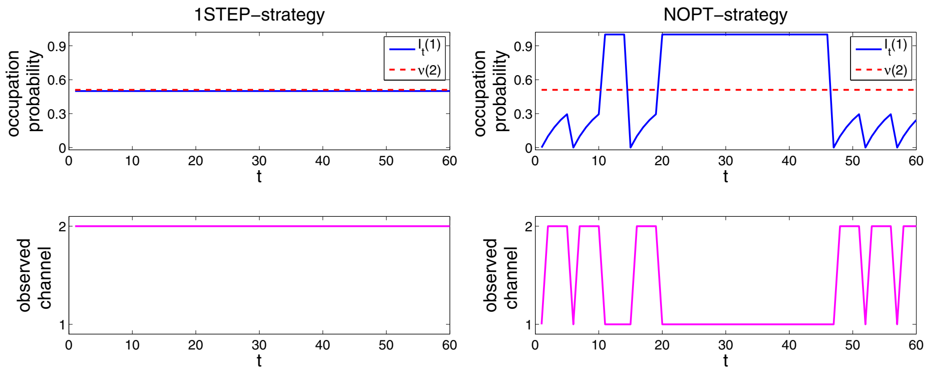


Fig. 3. Top: evolution of the first component of the internal state $I_t(1)$ (solid line) compared to the stationary probability of the second channel $\nu(2)$ (doted line); bottom: evolution of the observed channel; for the 1STEP-strategy (left plots) and the NOPT-strategy (right plots) in the two-channel model with $\alpha = (0.1, 0.51)'$ and $\beta = (0.9, 0.51)'$.

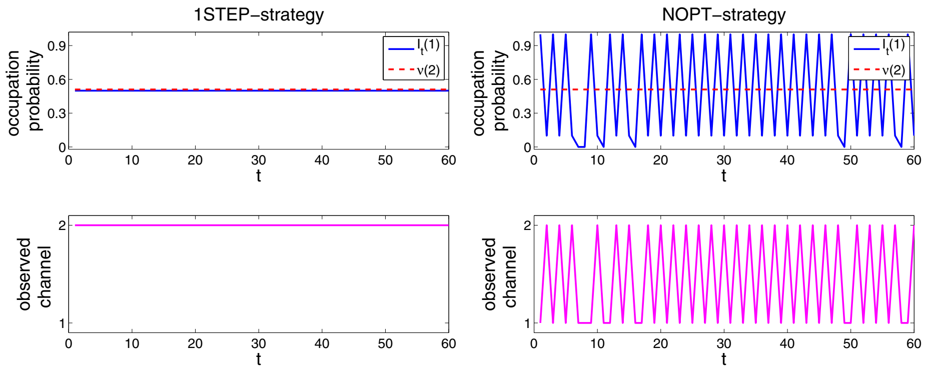


Fig. 4. Top: evolution of the first component of the internal state $I_t(1)$ (solid line) compared to the stationary probability of the second channel $\nu(2)$ (doted line); bottom: evolution of the observed channel; for the 1STEP-strategy (left plots) and the NOPT-strategy (right plots) in the two-channel model with $\alpha = (0.9, 0.51)'$ and $\beta = (0.1, 0.51)'$.

evolution of the probability $I_t(1)$ that the first channel is idle compared to the stationary probability of the second channel $\nu(2) = 0.51$. At the bottom, we represent the channel actually observed at each time according to the policy. In this first scenario, the stationary probability of the first channel ($\nu(1) = 0.5$) is lower than that of the second channel and the 1STEP-strategy always selects the second channel. However, using the knowledge that the first channel stays in the same state during long periods, the NOPT-strategy proposes a different choice of actions sometimes observing the first channel and continuing to observe it until it becomes busy.

In the second scenario, we use the same parameters except that the state of channel 1 fluctuates strongly i.e. $\alpha(1) = 1 - \beta(1) = 0.9$. We can observe in Fig. 4 that the results are similar to scenario 1 for the 1STEP-strategy and that the NOPT-strategy takes advantage of the fluctuating channel observing it when it is idle.

In scenario 3, we study the situation where channel 2 has a stationary probability smaller than channel 1. We use $\alpha = (0.1, 0.49)'$ and $\beta = (0.9, 0.49)'$. In this case, the 1STEP-strategy is quite interesting (see Fig. 5): when channel 1 is idle with probability 1, it is sensed and accessed, and, as soon as it has been seen to be occupied, channel 2 is observed while $I_t(1)$ is lower than $\nu(2) = 0.49$.

We simulate 200 trajectories of length 10000 and compute along each realization the average rewards obtained using the random choice (RAND-strategy), the 1STEP-strategy and the NOPT-strategy. We summarize these results for the three scenarios discussed above in Tab. 1. Remark that the mean of the average reward with the NOPT-strategy is always higher than the one with the 1STEP-strategy. For the first and the second scenarios, the average reward obtained using the 1STEP-strategy is just a little higher than the one obtained with the RAND-strategy which is around 0.5. This is not surprising since the

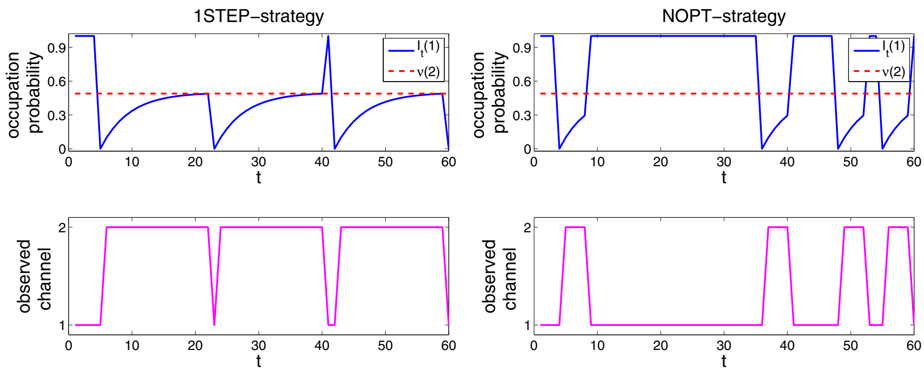


Fig. 5. Top: evolution of the first component of the internal state $I_t(1)$ (solid line) compared to the stationary probability of the second channel $\nu(2)$ (doted line); bottom: evolution of the observed channel; for the 1STEP-strategy (left plots) and the NOPT-strategy (right plots) in the two-channel model with $\alpha = (0.1, 0.49)'$ and $\beta = (0.9, 0.49)'$.

Table 1. The average reward obtained with the RAND-strategy, the 1STEP-strategy and the NOPT-strategy for different values of α and β. The interquartile range is in brackets.

	α	β	RAND-strategy	1STEP-strategy	NOPT-strategy
scenario 1	(0.1,0.51)	(0.9,0.51)	0.505 (0.01)	0.51 (0.07)	0.646 (0.01)
scenario 2	(0.9,0.51)	(0.1,0.51)	0.505 (0.06)	0.51 (0.07)	0.687 (0.06)
scenario 3	(0.1,0.49)	(0.9,0.49)	0.497 (0.01)	0.578 (0.12)	0.636 (0.14)

1STEP-strategy selects at each slot channel 2 which is idle with probability 0.51 so the mean of the average reward is near 0.51. In the third scenario, the average reward using 1STEP-strategy is better than in the scenarios 1 and 2 since the policy is more complicated.

In the three first scenarios, one channel is either persistent or very fluctuating and the NOPT-strategy takes advantage of these situations. When the transitions probabilities $\alpha(i)$ and $\beta(i)$ ($i = 1, 2$) are not close to 0 or 1, the NOPT-strategy is still preferable to the 1STEP-strategy but the perfomance gap between them is no very significant.

Finally, we observe the impact of the discretization parameter K on the NOPT-srategy and the discount parameter γ on both strategies. We display in Fig. 6 the average reward obtained in the first scenario using the NOPT-strategy with differents values of K. We observe that even when there are few internal points (here K lower than 5), the NOPT-strategy obtains good results. Discretizing the internal state space with K larger than 5 has no impact on the average reward. Note that for other values of transition probabilities α and β (in particular if $|\beta(i) - \alpha(i)|$ is very close to 1), it can be necessary to use more internal points.

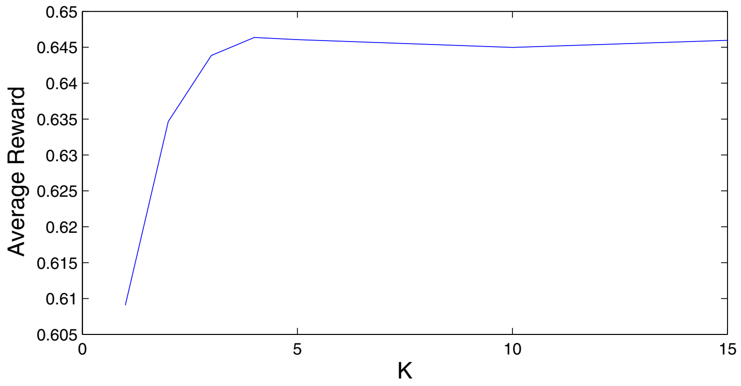


Fig. 6. Average reward for the NOPT-strategy depending on different values of K in a two-channel model with $\alpha = (0.1, 0.51)'$ and $\beta = (0.9, 0.51)'$.

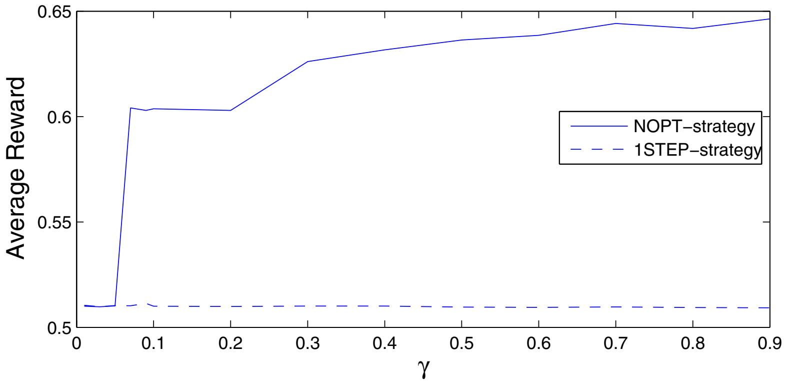


Fig. 7. Average reward for the 1STEP-strategy (doted line) and the NOPT-strategy (solid line) depending on different values of γ in a two-channel model with $\alpha = (0.1, 0.51)'$ and $\beta = (0.9, 0.51)'$.

In Fig. 7, we display the average reward obtained using the NOPT-strategy and the 1STEP-strategy. The average rewards of the NOPT-strategy logically increases with γ. The 1STEP-strategy is obviously not affected by this variation. When γ is low, the rewards obtained are the same with both strategies, which is consistent: a strategy maximizing a highly discounted expected reward then becomes close to a strategy maximizing the expected one-step reward.

5 Conclusion

We have presented an algorithm for computing a near optimal policy for channel allocation considered as an instance of the planning problem in a POMDP. Using the independence of the channels, a N-dimensional internal state differing from the standard belief state is computed and the POMDP is converted into a continuous state MDP. In order to obtain the optimal value function, we constructed a set of internal points where value iteration is used. The near optimal policy is then greedy with respect to the approximate optimal value function. The simulations results show that the proposed strategy is better than the previously proposed one-step strategy in a two-channel model when one channel is either persistent or very fluctuating. Similar gain also holds when there are three channels or more. However, the algorithmic complexity reduces the practicability of the proposed approach for more than ten channels. We are currently extending this approach to the case where the marginal distribution of the state transitions of the channels is not known beforehand.

References

1. Cassandra, A., Littman, M., Zhang, N., et al.: Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In: Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence, pp. 54–61 (1997)
2. Meuleau, N., Kim, K., Kaelbling, L., Cassandra, A.: Solving POMDPs by searching the space of finite policies. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 417–426 (1999)
3. Aberdeen, D.: Policy-Gradient Algorithms for Partially Observable Markov Decision Processes. Ph.D thesis, Australian National University (2003)
4. Pineau, J., Gordon, G., Thrun, S.: Anytime point-based approximations for large POMDPs. Journal of Artificial Intelligence Research 27, 335–380 (2006)
5. Astrom, K.: Optimal control of Markov decision processes with incomplete state estimation. Journal of Mathematical Analysis and Applications 10, 174–205 (1965)
6. Sondik, E.: The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon: Discounted Costs. Operations Research 26(2), 282–304 (1978)
7. Kaelbling, L., Littman, M., Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101(1), 99–134 (1996)
8. Zhao, Q., Tong, L., Swami, A.: Decentralized cognitive MAC for dynamic spectrum access. In: Proc. First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, pp. 224–232 (2007)
9. Whittle, P.: Restless Bandits: Activity Allocation in a Changing World. Journal of Applied Probability 25, 287–298 (1988)
10. Papadimitriou, C., Tsitsiklis, J.: The complexity of optimal queueing network control. In: Proceedings of the Ninth Annual Structure in Complexity Theory Conference, pp. 318–322 (1994)
11. Le Ny, J., Dahleh, M., Feron, E.: Multi-UAV Dynamic Routing with Partial Observations using Restless Bandits Allocation Indices. LIDS, Massachusetts Institute of Technology, Tech. Rep. (2007)
12. Guha, S., Munagala, K.: Approximation Algorithms for Partial-Information Based Stochastic Control with Markovian Rewards. In: 48th Annual IEEE Symposium on Foundations of Computer Science, 2007. FOCS 2007, pp. 483–493 (2007)
13. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York (1994)
14. Bonet, B.: An e-optimal grid-based algorithm for partially observable Markov decision processes. In: 19th International Conference on Machine Learning, Sydney, Australia (June 2002)

Evaluation of Batch-Mode Reinforcement Learning Methods for Solving DEC-MDPs with Changing Action Sets

Thomas Gabel and Martin Riedmiller

Neuroinformatics Group
Department of Mathematics and Computer Science
University of Osnabrück, 49069 Osnabrück, Germany
{thomas.gabel,martin.riedmiller}@uni-osnabrueck.de

Abstract. DEC-MDPs with changing action sets and partially ordered transition dependencies have recently been suggested as a sub-class of general DEC-MDPs that features provably lower complexity. In this paper, we investigate the usability of a coordinated batch-mode reinforcement learning algorithm for this class of distributed problems. Our agents acquire their local policies independent of the other agents by repeated interaction with the DEC-MDP and concurrent evolvement of their policies, where the learning approach employed builds upon a specialized variant of a neural fitted Q iteration algorithm, enhanced for use in multi-agent settings. We applied our learning approach to various scheduling benchmark problems and obtained encouraging results that show that problems of current standards of difficulty can very well approximately, and in some cases optimally be solved.

1 Introduction

Decentralized decision-making is required in many real-life applications. Examples include distributed sensor networks, teams of autonomous robots, rescue operations where units must decide independently which sites to search, or production planning and factory optimization where machines may act independently with the goal of achieving optimal joint productivity. The interest in analyzing and solving decentralized learning problems is to a large degree evoked by their high relevance for practical problems. While Markov decision processes (MDP) have proven to be a suitable tool for solving problems involving a single agent, a number of extensions of these models to multi-agent systems have been suggested. Among those, the DEC-MDP framework [4], that is characterized by each agent having only a partial view of the global system state, has been frequently investigated. It has been shown that the complexity of general DEC-MDPs is NEXP-complete, even for the benign case of two cooperative agents [4].

The enormous computational complexity of solving DEC-MDPs conflicts with the fact that real-world tasks do typically have a considerable problem size. Taking this into consideration, we recently [10] identified a subclass of general DEC-MDPs that features regularities in the way the agents interact with one another.

For this class, we could show that the complexity of optimally solving an instance of such a DEC-MDP is provably lower (NP-complete) than the general problem.

In this paper, we focus on job-shop scheduling problems which can be modelled using the DEC-MDP class mentioned above. Since such problems involve settings with ten and more agents, optimal solution methods can hardly be applied. Therefore, we propose for employing a multi-agent reinforcement learning approach, where the agents are independent learners and do their learning online. The disadvantage of choosing this learning approach is that agents may take potentially rather bad decisions until they learn better ones and that, hence, only an approximate joint policy may be obtained. The advantage is, however, that the entire learning process is done in a completely distributed manner with each agent deciding on its own local action based on its partial view of the world state and on any other information it eventually gets from its teammates.

In Section 2, we summarize and illustrate the key properties of the class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies [10], which are in the center of our interest. Section 3 discusses a method that allows for partially resolving some of the inter-agent dependencies. Subsequently (Section 4), we provide the basics of our learning approach to acquire approximate joint policies using coordinated multi-agent reinforcement learning. Finally, in Section 5 we show how scheduling problems can be modelled using the class of DEC-MDPs specified. Moreover, empirical results for solving various scheduling benchmark problems are presented.

2 Decentralized MDPs

The subclass of problems we are focusing on may feature an arbitrary number of agents whose actions influence, besides their own, the state transitions of maximally one other agent in a specific manner. Formally defining the problem settings of our interest, we embed them into the framework of decentralized Markov decision processes (DEC-MDP) by Bernstein et al. [4].

Definition 1. *A factored m-agent DEC-MDP M is defined by a tuple*

$$\langle Ag, S, A, P, R, \Omega, O \rangle \text{ with}$$

– $Ag = \{1, \dots, m\}$ *as the set of agents,*
– S *as the set of world states which can be factored into m components $S = S_1 \times \cdots \times S_m$ (the S_i belong to one of the agents each),*
– $A = A_1 \times ... \times A_m$ *as the set of joint actions to be performed by the agents ($a = (a_1, \dots, a_m) \in A$ denotes a joint action that is made up of elementary actions a_i taken by agent i),*
– P *as the transition function with $P(s'|s, a)$ denoting the probability that the system arrives at state s' upon executing a in s,*
– R *as the reward function with $R(s, a, s')$ denoting the reward for executing a in s and transitioning to s',*
– $\Omega = \Omega_1 \times \cdots \times \Omega_m$ *as the set of all observations of all agents ($o = (o_1, \dots, o_m) \in \Omega$ denotes a joint observation with o_i as the observation for agent i),*

– *O as the observation function that determines the probability $O(o_1, \dots, o_m | s, a, s')$ that agent 1 through m perceive observations o_1 through o_m upon the execution of a in s and entering s'.*
– *M is jointly fully observable, the current state is fully determined by the amalgamation of all agents' observations: $O(o|s, a, s') > 0 \Rightarrow Pr(s'|o) = 1$.*

We refer to the agent-specific components $s_i \in S_i$, $a_i \in A_i$, $o_i \in \Omega_i$ as local state, action, and observation of agent i, respectively. A joint policy π is a set of local policies $\langle \pi_1, \dots, \pi_m \rangle$ each of which is a mapping from agent i's sequence of local observations to local actions, i.e. $\pi_i : \overline{\Omega}_i \to A_i$. Simplifying subsequent considerations, we may allow each agent to fully observe its local state.

Definition 2. *A factored m-agent DEC-MDP has local full observability, if for all agents i and for all local observations o_i there is a local state s_i such that $Pr(s_i|o_i) = 1$.*

Note that joint full observability and local full observability of a DEC-MDP do generally not imply full observability, which would allow us to consider the system as a single large MDP and to solve it with a centralized approach. Instead, typically vast parts of the global state are hidden from each of the agents.

A factored m-agent DEC-MDP is called *reward independent*, if there exist local functions R_1 through R_m, each depending on local states and actions of the agents only, as well as a function r that amalgamates the global reward value from the local ones, such that maximizing each R_i individually also yields a maximization of r. If, in a factored m-agent DEC-MDP, the observation each agent sees depends only on its current and next local state and on its action, then the corresponding DEC-MDP is called *observation independent*, i.e. $P(o_i|s, a, s', (o_1 \dots o_{i-1}, o_{i+1} \dots o_m) = P(o_i|s_i, a_i, s'_i)$. Then, in combination with local full observability, the observation-related components Ω and O are redundant and can be removed from Definition 1.

While the DEC-MDPs of our interest are observation independent and reward independent, they are *not* transition independent. That is, the state transition probabilities of one agent may very well be influenced by another agent. However, we assume that there are some regularities, to be discussed in the next section, that determine the way local actions exert influence on other agents' states.

2.1 Variable Action Sets

The following two definitions characterize the specific subclass of DEC-MDPs we are interested in. Firstly, we assume that the sets of local actions A_i change over time.

Definition 3. *An m-agent DEC-MDP with factored state space $S = S_1 \times \dots \times S_m$ is said to feature* changing action sets*, if the local state of agent i is fully described by the set of actions currently selectable by that agent ($s_i = A_i \setminus \{\alpha_0\}$) and A_i is a subset of the set of all available local actions $\mathcal{A}_i = \{\alpha_0, \alpha_{i1} \dots \alpha_{ik}\}$, thus $S_i = \mathcal{P}(\mathcal{A}_i \setminus \{\alpha_0\})$. Here, α_0 represents a null action that does not change the state and is always in A_i. Subsequently, we abbreviate $\mathcal{A}_i^r = \mathcal{A}_i \setminus \{\alpha_0\}$.*

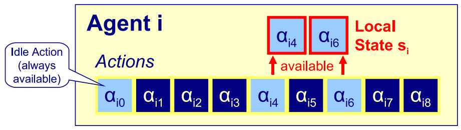

Agent i
Actions
Idle Action (always available)
Local State s_i
available
α_{i0} α_{i1} α_{i2} α_{i3} α_{i4} α_{i5} α_{i6} α_{i7} α_{i8}
α_{i4} α_{i6}

Fig. 1. DEC-MDPs with Changing Action Sets: Local State of Agent i

Concerning state transition dependencies, one can distinguish between dependent and independent local actions. While the former influence an agent's local state only, the latter may additionally influence the state transitions of other agents. As pointed out, our interest is in non-transition independent scenarios. In particular, we assume that an agent's local state can be affected by an arbitrary number of other agents, but that an agent's local action affects the local state of maximally one other agent.

Definition 4. *A factored m-agent DEC-MDP has* partially ordered transition dependencies, *if there exist dependency functions σ_i for each agent i with*

1. $\sigma_i : \mathcal{A}_i^r \rightarrow Ag \cup \{\emptyset\}$ *and*
2. $\forall \alpha \in \mathcal{A}_i^r$ *the directed graph* $G_\alpha = (Ag \cup \{\emptyset\}, E)$ *with* $E = \{(j, \sigma_j(\alpha)) | j \in Ag\}$ *is acyclic and contains only one directed path*

and it holds

$$\begin{aligned} & P(s_i'|s, (a_1 \ldots a_m), (s_1' \ldots s_{i-1}', s_{i+1}' \ldots s_m')) \\ = \; & P(s_i'|s_i, a_i, \{a_j \in \mathcal{A}_j | i = \sigma_j(a_j), j \neq i\}) \end{aligned}$$

The influence exerted on another agent always yields an extension of that agent's action set: If $\sigma_i(\alpha) = j$, i takes local action α, and the execution of α has been finished, then α is added to $A_j(s_j)$, while it is removed from $A_i(s_i)$.

That is, the dependency functions σ_i indicate whose other agents' states are affected when agent i takes a local action. Further, Definition 4 implies that for each local action α there is a total ordering of its execution by the agents. While these orders are total, the global order in which actions are executed is only partially defined by that definition and subject to the agents' policies.

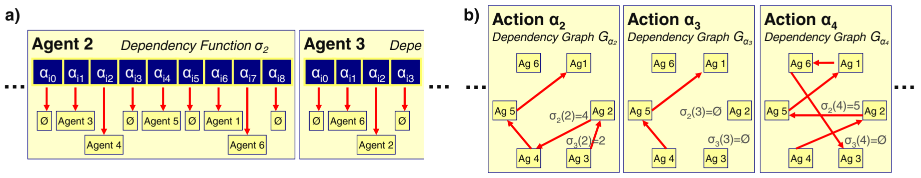

a)
Agent 2
Dependency Function σ_2
α_{i0} α_{i1} α_{i2} α_{i3} α_{i4} α_{i5} α_{i6} α_{i7} α_{i8}
Ø Agent 3 Ø Agent 5 Ø Agent 1 Ø
Agent 4 Agent 6
Agent 3
Depe
α_{i0} α_{i1} α_{i2} α_{i3}
Ø Agent 6 Ø
Agent 2
b)
Action α_2
Dependency Graph G_{α_2}
Ag 6 Ag1 Ag 5 Ag 2 Ag 4 Ag 3
$\sigma_2(2)=4$
$\sigma_3(2)=2$
Action α_3
Dependency Graph G_{α_3}
Ag 6 Ag 1 Ag 5 Ag 2 Ag 4 Ag 3
$\sigma_2(3)=Ø$
$\sigma_3(3)=Ø$
Action α_4
Dependency Graph G_{α_4}
Ag 6 Ag 1 Ag 5 Ag 2 Ag 4 Ag 3
$\sigma_2(4)=5$
$\sigma_3(4)=Ø$

Fig. 2. Exemplary Dependency Functions (a) and Graphs (b)

In [10] it is shown that, for the class of problems considered, any local action may appear only once in an agent's action set and, thus, may be executed only once. Further, it is proved that solving a factored m-agent DEC-MDP with changing action sets and partially ordered dependencies is NP-complete.

3 Reactive Policies and Resolved Dependencies

An agent that takes its action based solely on its most recent local observation $s_i \subseteq \mathcal{A}_i$ will in general not be able to contribute to optimal joint behavior. In particular, it will have difficulties in assessing the value of taking its idle action α_0. Taking α_0, the local state remains unchanged except when it is influenced by dependent actions of other agents.

Definition 5. *For a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies, a* reactive policy $\pi^r = \langle \pi_1^r \dots \pi_m^r \rangle$ *consists of m reactive local policies with $\pi_i^r : S_i \rightarrow \mathcal{A}_i^r$ where $S_i = \mathcal{P}(\mathcal{A}_i^r)$.*

So, purely reactive policies always take an action $\alpha \in \mathcal{A}_i(s_i) = s_i$ (except for $s_i = \emptyset$), even if it was more advisable to stay idle and wait for a transition from s_i to some $s_i' = s_i \cup \{\alpha'\}$ induced by another agent, and then execute α' in s_i'.

3.1 Communication-Based Awareness of Dependencies

The probability that agent i's local state moves to s_i' depends on three factors: on that agent's current local state s_i, on its action a_i, as well as on the set $\{a_j \in \mathcal{A}_j | i = \sigma_j(a_j), i \neq j\} = \varDelta_i$, i.e. on the local actions of all agents that may influence agent i's state transition. Let us for the moment assume that agent i always knows the set $\varDelta_i$. Then, all transition dependencies would be resolved as they would be known to each agent. As a consequence, all local transitions would be Markovian and local states would represent a sufficient statistic for each agent to behave optimally.

Unfortunately, fulfilling the assumption of all $\varDelta_i$ to be known conflicts with the idea of decentralized decision-making. In fact, knowing σ_j and relevant actions a_j of other agents, enables agent i to determine their influence on its local successor state and to best select its local action a_i. This action, however, generally also influences another agent's transition and, hence, that agent's action choice if it knows its set $\varDelta_j$, as well. Thus, it can be seen that even in the benign case of a two-agent system, there may be circular dependencies, which is why knowing all $\varDelta_i$ entirely would only be possible if a central decision-maker employing a joint policy and deciding for joint actions is used.

Nevertheless, we may enhance the capabilities of a reactive agent i by allowing it to get at least some partial information about $\varDelta_i$. For this, we extend a reactive agent's local state space from $S_i = \mathcal{P}(\mathcal{A}_i^r)$ to $\hat{S}_i$ such that for all $\hat{s}_i \in \hat{S}_i$ it holds $\hat{s}_i = (s_i, z_i)$ with $z_i \in \mathcal{P}(\mathcal{A}_i^r \setminus s_i)$. So, z_i is a subset of the set of actions currently *not* in the action set of agent i.

Definition 6. *Let $1 \dots m$ be reactive agents acting in a DEC-MDP, as specified in Definition 4, whose local state spaces are extended to $\hat{S}_i$. Assume that current local actions $a_1 \dots a_m$ are taken* consecutively. *Given that agent j decides for $a_j \in A_j(s_j)$ and $\sigma_j(a_j) = i$, let also s_i be the local state of i and $\hat{s}_i$ its current extended local state with $\hat{s}_i = (s_i, z_i)$. Then, the transition dependency between j and i is said to be* resolved, *if $z_i := z_i \cup \{a_j\}$.*

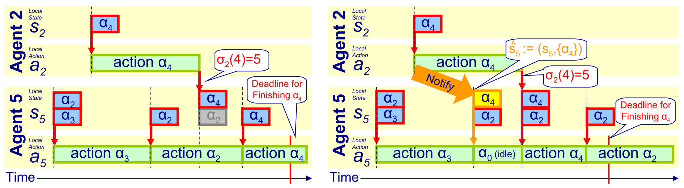


Fig. 3. Left: Agent 5 behaves purely reactively. Right: A notification from agent 2 allows for resolving a dependency, agent 5 may stay willingly idle and meet its deadline.

The resolution of a transition dependency according to Definition 6 corresponds to letting agent i know some of those current local actions of other agents by which the local state of i will soon be influenced. Because, for the class of problems we are dealing with, inter-agent interferences are always exerted by changing (extending) another agent's action set, agent i gets to know which further action(s) will soon be available in its action set. By integrating this piece of information into i's extended local state description $\hat{S}_i$, this agent obtains the opportunity to willingly stay idle (execute α_0) until the announced action $a_j \in z_i$ enters its action set and can finally be executed (see Figure 3 for an example). Thus, because local states $\hat{s}_i$ are extended by information relating to transition dependencies between agents, such policies are normally more capable than purely reactive ones, since at least some information about future local state transitions induced by teammates can be regarded during decision-making.

The notification of agent i, which instructs it to extend its local state component z_i by a_j, may easily be realized by a simple message passing scheme (assuming cost-free communication between agents) that allows agent i to send a single directed message to agent $\sigma_i(\alpha)$ upon the local execution of α.

4 Policy Acquisition with Reinforcement Learning

Solving a DEC-MDP optimally is NEXP-hard and intractable for all except the smallest problem sizes. Unfortunately, the fact that the subclass of DEC-MDPs we identified in Section 2 is in NP and hence simpler to solve, does not rid us from the computational burden implied. Given that fact, our goal is not to develop yet another optimal solution algorithm that is applicable to small problems only, but to look for a technique capable of quickly obtaining approximate solutions in the vicinity of the optimum.

Reinforcement learning (RL) has proven to be usable for acquiring approximate policies in decentralized MDPs. In contrast to offline planning algorithms, RL allows for a real decentralization of the problem employing independently learning agents. However, due to inter-agent dependencies designing distributed learning algorithms represents a challenging task.

In the remainder of this section, we outline the basic characteristics of our approach to applying RL in distributed settings aiming at the acquisition of joint policies for m-agent factored DEC-MDPs with changing action sets.

4.1 Challenges for Independent Learners

Boutilier [5] pointed out that any multi-agent system can be considered as a single MDP when adopting an external point of view. The difficulties induced when taking the step towards decentralization can be grouped into three categories. First, in addition to the (single-agent) temporal credit assignment problem, the multi-agent credit assignment problem arises, which corresponds to answering the question of whose agent's local action contributed how much to a corporate success. To this end, we consider reward independent DEC-MDPs only (see Section 2) with the global reward being the sum of local ones.

A second challenge is represented by the agents' uncertainty regarding the other agents' policies pursued during learning. To sidestep that problem, we revert to an inter-agent coordination mechanism introduced in [12]. Here, the basic idea is that each agent always optimistically assumes that all other agents behave optimally (though they often will not, e.g. due to exploration). Updates to the value function and policy learned are only done when an agent is certain that a superior joint action has been executed. Since the performance of that coordination scheme quickly degrades in the presence of noise, we focus on deterministic DEC-MDPs in the remainder of the paper.

Third, the subclass of DEC-MDPs identified in Section 2 has factored state spaces providing each agent with (locally fully observable) state perceptions. Since the global state is unknown, each agent must necessarily remember the full history of local states to behave optimally, which quickly becomes intractable even for toy problems (see [10] for our alternative approach of compactly encoding the agents' state histories). In Section 3.1 we have suggested a message passing scheme that enables the learners to inform other agents about expected state transitions and thus enhances the capabilities of a purely reactive agent. Although, in this way the optimal policy can generally not be represented, the need for storing full state histories can be avoided.

4.2 Joint Policy Acquisition with Reinforcement Learning

We let the agents acquire their local policies independently of the other agents by repeated interaction with the DEC-MDP and concurrent evolvement of their policies. Our learning approach is made up of alternating data collection and learning stages that are being run concurrently within all agents. At its core, a neural fitted Q iteration (NFQ) algorithm [14] is used that allows the agents to determine a value function over their local state-action spaces.

4.2.1 Data Collection

Our multi-agent extension of NFQ denotes a batch-mode RL algorithm where agent i computes an approximation of the optimal policy, given a finite set $\mathbb{T}_i$ of local four-tuples [8]. $\mathbb{T}_i = \{(s_i^k, a_i^k, r_i^k, s_i^{'k}) | k = 1 \dots p\}$ can be collected in any arbitrary manner (e.g. by an ϵ-greedy policy) and contains agent-specific local states s_i^k, local actions $a_i^k \in A_i(s_i^k) = s_i^k \subseteq \mathcal{A}_i$, corresponding rewards r_i^k, as well as local successor states $s_i^{'k}$ entered. If the final state of the DEC-MDP has been reached ($A_i(s_i) = \emptyset$ for all i), the system is reset to its starting state (beginning of what we call a new training episode), and if a sufficient amount of tuples has been collected, the learning stage (4.2.2) is entered.

4.2.2 Applying Neural Fitted Q Iteration

Given $\mathbb{T}_i$ and a regression algorithm, NFQ iteratively computes an approximation $\tilde{Q}_i : S_i \times \mathcal{A}_i \rightarrow \mathbb{R}$ of the optimal state-action value function, from which a policy $\tilde{\pi}_i : S_i \rightarrow \mathcal{A}_i$ can be induced by greedy exploitation via $\tilde{\pi}_i(s_i) = \arg\max_{\alpha \in A_i(s_i)} \tilde{Q}_i(s_i, \alpha)$. Having initialized $\tilde{Q}_i$ and counter q to zero, NFQ repeatedly processes the following steps until some stop criterion becomes true:

1. construct training set $\mathbb{F}_i$ as input for the regression algorithm according to $\mathbb{F}_i = \{(v^k, w^k) | k = 1 \dots p\}$, with $v^k = (s_i^k, a_i^k)$, target values w^k are calculated using the Q learning [18] update rule, $w^k = r_i^k + \gamma \max_{\alpha \in s_i^k} \tilde{Q}_i^q(s_i'^k, \alpha)$,
2. use the regression algorithm and $\mathbb{F}_i$ to induce a new approximation $\tilde{Q}_i^{q+1} : S_i \times \mathcal{A}_i \rightarrow \mathbb{R}$, and increment q.

For the second step, NFQ employs multi-layer perceptron neural networks in conjunction with the efficient backpropagation variant Rprop [15].

4.2.3 Optimistic Inter-agent Coordination

For the multi-agent case, we modify step 2 of applying NFQ: Agent i creates a reduced (optimistic) training set $\mathbb{O}_i$ such that $|\mathbb{O}_i| \leq |\mathbb{F}_i|$. Given a deterministic environment and the resetting mechanism during data collection (4.2.1), the probability that agent i enters some s_i^k more than once is larger than zero. Hence, if a certain action $a_i^k \in A_i(s_i^k)$ has been taken multiple times in s_i^k, it may—because of differing local actions selected by other agents—have yielded very different rewards and local successor states for i. Instead of considering all tuples from $\mathbb{T}_i$, only those are used for creating $\mathbb{O}_i$ that have resulted in maximal expected rewards. This means, we assume that all other agents take their best possible local action, which are, when combined with a_i^k, most suitable for the current global state. Accordingly, we compute the optimistic target values w^k for a given local state-action pair $v^k = (s_i^k, a_i^k)$ according to

$$w^k := \max_{\substack{(s_i^k, a_i^k, r_i^k, s_i'^k) \in \mathbb{T}_i, \\ (s_i^k, a_i^k) = v^k}} \left(r_i^k + \gamma \max_{\alpha \in s_i^k} \tilde{Q}_i^k(s_i'^k, \alpha) \right)$$

Consequently, $\mathbb{O}_i$ realizes a partitioning of $\mathbb{T}_i$ with respect to identical values of s_i^k and a_i^k, and w^k is the maximal sum of the immediate rewards and discounted expected costs over all tuples $(s_i^k, a_i^k, \cdot, \cdot) \in \mathbb{T}_i$.

5 Experiments

Distributed problem solving often faces situations where a larger number of agents are involved and where a factored system state description is given with the agents taking their decisions based on local observations. Also, our assumptions that local actions may influence the state transitions of maximally one other agent and that any action has to be performed only once are frequently fulfilled. Sample real-world applications include scenarios from manufacturing, traffic control, or assembly line optimization, where typically the production of a good involves a number of processing steps that have to be performed in a specific order. In a factory, however, usually a variety of products is assembled concurrently, which is why an appropriate sequencing and scheduling of single operations is of crucial importance for overall performance. Our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies covers a variety of such scheduling problems, for example flow-shop and job-shop scheduling scenarios [13], even scheduling problems with recirculating tasks can be modelled. Next, we show how our class of DEC-MDPs can be utilized for modeling production planning problems and evaluate the performance of our learning approach using a variety of established benchmarks.

5.1 Scheduling Problems

The goal of scheduling is to allocate a specified number of jobs to a limited number of resources (also called machines) such that some objective is optimized. In job-shop scheduling (JSS), n jobs must be processed on m machines in a pre-determined order. Each job j consists of ν_j operations $o_{j,1} \dots o_{j,\nu_j}$ that have to be handled on a certain resource $\varrho(o_{j,k})$ for a specific duration $\delta(o_{j,k})$. A job is finished after its last operation has been entirely processed (completion time f_j). In general, scheduling objectives to be optimized all relate to the completion time of the jobs. In this paper, we concentrate on the goal of minimizing maximum makespan ($C_{max} = max_j\{f_j\}$), which corresponds to finishing processing as quickly as possible.

Solving JSS problems is well-known to be NP-hard. Over the years, numerous benchmark problem instances of varying sizes have been established, a collection of sample problems is provided by the OR Library [1]. A common characteristic of those JSS benchmarks is that usually no recirculation of jobs is allowed, i.e. that each job must be processed exactly once on each resource ($\nu_j = m$). For more basics on scheduling, the reader is referred to [13].

JSS problems can be modelled using factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies:

– The world state can be factored: To each of the resources one agent i is associated whose local action is to decide which waiting job to process next.
– The local state of i can be fully described by the changing set of jobs currently waiting for further processing. Since choosing and executing a job represents a local action (i.e. $\mathcal{A}_i^r$ is the set of jobs that must be processed on resource i), it holds $S_i = \mathcal{P}(\mathcal{A}_i^r)$.
– After having finished an operation of a job, this job is transferred to another resource, which corresponds to influencing another agent's local state by extending that agent's action set.
– The order of resources on which a job's operation must be processed is given in a JSS problem. Therefore, we can define $\sigma_i : \mathcal{A}_i^r \rightarrow Ag \cup \{\emptyset\}$ (cf. Definition 4) for all agents/resources i as

$$\sigma_i(\alpha) = \begin{cases} \emptyset & \text{if } k = \nu_\alpha \\ \varrho(o_{\alpha,k+1}) & \text{else} \end{cases}$$

where k corresponds to the number of that operation within job α that has to be processed on resource i, i.e. k such that $\varrho(o_{\alpha,k}) = i$.
– Given the no recirculation property from above and the definition of σ_i, the directed graph G_α from Definition 4 is indeed acyclic with one directed path.

More low-level details on solving JSS problems in a decentralized manner as well as on parameter settings of the RL algorithm involved, can be found in [9].

5.2 Experiment Outline

Classically, JSS problems are solved in a centralized manner, assuming that a central control over the process can be established. From a certain problem size on, however, the NP-hardness of the problem precludes the search for an optimal solution even for a centralized approach. That is why frequently dispatching priority rules are employed that take local dispatching decisions in a reactive and independent manner (the FIFO rule is a well-known example).

In the following experiment, however, a comparison to alternative scheduling methods is only our secondary concern. For comparison, we just provide results for two of the best-performing priority rules (SPT chooses operations with shortest processing time δ next and AMCC makes use of knowing the global system state), as well as the theoretic optimum, representing a lower bound, as it may be found by a centralized brute-force search. Our primary concern is on analyzing the following three approaches. We compare agents that independently learn

– purely reactive policies π_i^r (see Section 3) defined over $S_i = \mathcal{P}(\mathcal{A}_i^r)$ that never remain idle when their action set is not empty [**RCT**],
– reactive policies $\hat{\pi}_i$ that are partially aware of their dependencies on other agents (notified about forthcoming influences exerted by other agents) [**COM**],
– policies $\pi_i : E_i \rightarrow \mathcal{A}_i$ using full information about the agents' histories, here E_i is a compact encoding of that agent i's observation history $\overline{S}_i$ (see [10] for more details) [**ENC**].

In JSS problems, it typically holds that $d(o_{j,k}) > 1$ for all j and k. Since most of such durations are not identical, decision-making usually proceeds asynchronously across agents. We assume that a COM-agent i sends a message to agent $\sigma_i(\alpha)$ when it starts the execution of an operation from job α, announcing to that agent the arrival of α, whereas the actual influence on agent $\sigma_i(\alpha)$ (its action set extension) occurs $d(o_{\alpha,\cdot})$ steps later (after $o_{\alpha,\cdot}$ has been finished).

Classes of Schedules

For a problem with m resources and n jobs consisting of m operations each, there are $(n!)^m$ possible schedules (also called set of active schedules, $\mathbb{S}_a$). Considering such a problem as a DEC-MDP, this gives rise to, for example, about $1.4 \cdot 10^{17}$ possible joint policies for $m = n = 6$.

Considering purely reactive agents, the number of policies/schedules that can be represented is usually dramatically reduced. Unfortunately, only schedules from the class of non-delay schedules $\mathbb{S}_{nd}$ can be created by applying reactive policies. Since $\mathbb{S}_{nd} \subseteq \mathbb{S}_a$ and because it is known that the optimal schedule is always in $\mathbb{S}_a$ [13], but not necessarily in $\mathbb{S}_{nd}$, RCT-agents can at best learn the optimal solution from $\mathbb{S}_{nd}$. By contrast, learning with ENC-agents, in principle the optimal solution can be attained, but we expect that the time required by our learning approach for this to happen will increase significantly.

We hypothesize that the awareness of inter-agent dependencies achieved by partial dependency resolutions via communication may in fact realize a good trade-off between the former two approaches. On the one hand, when resolving a transition dependency according to Definition 6, an agent i can become aware of an incoming job. Thus, i may decide to wait for that arrival, instead of starting to execute another job. Hence, also schedules can be created that are not non-delay. On the other hand, very poor policies with unnecessary idle times can be avoided, since a decision to stay idle will be taken very dedicatedly, viz only when a future job arrival has been announced. This falls into place with the fact that the extension of an agent's local state to $\hat{s_i} = (s_i, z_i)$ is rather limited and consequently the number of local states is only slightly increased.

5.3 Illustrative Benchmark

We start off with the FT6 benchmark problem taken from [1]. This depicts a problem with 6 resources and 6 jobs consisting of 6 operations each, hence we consider a DEC-MDP with 6 independently learning agents. Figure 4 summarizes the learning curves for the three approaches we want to compare (note that the SPT/FIFO/AMCC rules yield $C_{max} = 88/77/55$, here, and are not drawn for clarity). Results are averaged over 10 experiment repetitions and indicators for best/worst runs are provided.

First of all, this experiment shows the effectiveness of our approach, since each type of learning agents considered manages to attain its respective optimum and because static dispatching rules with a local view are clearly outperformed. The FT6 benchmark is a problem, where the best reactive policy (hence, the best

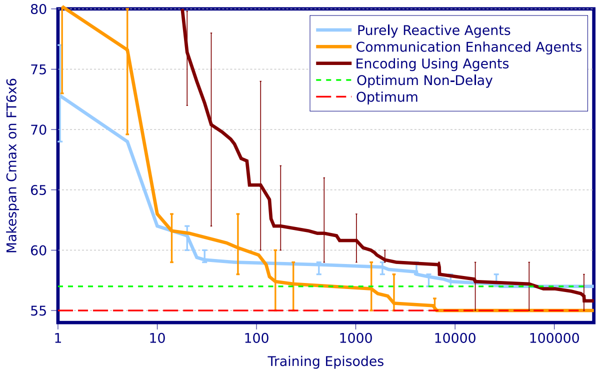


Fig. 4. Learning Curves for the FT6 Benchmark

Table 1. Learning results for scheduling benchmarks of varying size. All entries are average makespan values. The last column shows the relative remaining error (%) of the COM-agents compared to the theoretical optimum. Indices a, b, and c stand for problem sets provided by different authors.

$m \times n$	#Prbl	SPT	AMCC	Opt.	RCT	COM	ENC	Err
$5x10$	3	734.7	702.7	614.0	648.7	642.0	648.3	4.6
$10x10_a$	3	1174.3	1096.0	1035.7	1078.0	1062.7	1109.0	2.6
$10x10_b$	5	1000.2	894.2	864.2	899.0	894.6	928.6	3.5
$10x10_c$	9	1142.6	977.1	898.2	962.7	951.0	988.4	5.9
$5x20$	1	1267.0	1338.0	1165.0	1235.0	1183.0	1244.0	1.5
$15x20$	3	888.3	771.0	676.0	747.7	733.7	818.0	8.6

non-delay schedule with $C_{max} = 57$) is dragging behind, since the optimal solution corresponds to a delay schedule with makespan of 55. The steepest learning curve emerges for purely reactive agents that achieve the best non-delay solution, hence little interaction with the process is required for those agents to obtain high-quality policies. By contrast, ENC- and COM-agents are capable of learning the optimal policy, where the former require significantly more training time than the latter (note the log scale in Figure 4). This can be tributed to the clearly increased number of local states of ENC-agents, which have to cover the agents' state histories, and to the fact that they may take idle actions in principle in any state, while COM-agents do so only when a notification regarding forthcoming externally influenced state transitions has been received.

5.4 Benchmark Results

We also applied our framework to a large variety of different-sized benchmarks from [1] involving up to 15 agents and 20 jobs. In 12 out of the 37 benchmarks examined already the RCT version of our learning agents succeeded in acquiring the optimal joint policy. This also means that in those scenarios (all of them involved 5 resources) the optimal schedule is a non-delay one and we omit experiments using ENC- or COM-agent as no further improvement is possible.

Table 1 provides an overview of the results for the remaining, more intricate 25 benchmark problems (except for FT6, cf. Section 5.3), grouped by problem sizes ($m \times n$). This summary gives the quality of policies obtained after 25000 training episodes. Since ENC-agents have shown to require substantially longer to acquire high-quality policies, the results in the corresponding column are expectedly poor. However, while purely reactive agents already outperform standard rules, their enhancement by means of dedicated communication yields excellent improvements in all cases.

6 Related Work

One of the first formal approaches to model cooperative multi-agent systems was the MMDP framework by Boutilier [5], which requires every agent to be aware of the current global state. By contrast, factored state information including local partial/full observability are key ingredients of the DEC-POMDP framework of Bernstein et al. [4]. While the general problem has NEXP-complete complexity, other researchers have subsequently identified specific subclasses with lower computational complexity, e.g. transition independent DEC-MDPs [3] and DEC-MDPs with synchronizing communication [11]. While these subclasses are quite distinct, our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies features some commonalities with DEC-MDPs with event-driven interactions [2] where the latter focus on systems with two agents only and assume less structure in the inter-agent dependencies.

Independently learning agents have been targeted in a number of recent publications, e.g. [6,7]. Communication as a means of conveying information that is local to one agent to others has been investigated, for instance, in [11]. Here, policy computation is facilitated by allowing agents to fully synchronize their local histories of observations. By contrast, in the paper at hand we have explored a very limited form of directed communication that informs other agents about forthcoming interferences on state transition. Other approaches with limited communication can be found in [16] where each agent broadcasts its expected gain of a learning update and coordination is realized by performing collective learning updates only when the sum of the gains for the team as a whole is positive, or in [17] where communication is employed to enable a coordinated multi-agent exploration mechanism.

7 Conclusion

Decentralized Markov decision processes with changing action sets and partially ordered transition dependencies have been suggested as a sub-class of general DEC-MDPs that features provably lower complexity. In this paper, we have explored the usability of a coordinated batch-mode reinforcement learning algorithm for this class of distributed problems, that facilitates the agents to concurrently and independently learn their local policies of action. Furthermore, we have looked at possibilities for modeling memoryless agents and enhancing them by restricted allowance of communication.

The subclass of DEC-MDPs considered covers a wide range of practical problems. We applied our learning approach to production planning problems and evaluated it using numerous job-shop scheduling benchmarks that are already NP-hard when solved in a centralized manner. The results obtained are convincing insofar that benchmark problems of current standards of difficulty can very well be approximately solved by the learning method we suggest. The policies our agents acquire clearly surpass traditional dispatching rules and, in some cases, are able to solve the problem instances optimally.

Acknowledgements. This research has been supported by the German Research Foundation (DFG) under grant number Ri 923/2-3.

References

1. Beasley, J.: OR-Library (2005), http://people.brunel.ac.uk/~mastjjb/jeb/info.html
2. Becker, R., Zilberstein, S., Lesser, V.: Decentralized Markov Decision Processes with Event-Driven Interactions. In: Proceedings of AAMAS 2004, pp. 302–309. ACM Press, New York (2004)
3. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.: Solving Transition Independent Decentralized MDPs. Journal of AI Research 22, 423–455 (2004)
4. Bernstein, D., Givan, D., Immerman, N., Zilberstein, S.: The Complexity of Decentralized Control of Markov Decision Processes. Mathematics of Operations Research 27(4), 819–840 (2002)
5. C.: Sequential Optimality and Coordination in Multiagent Systems. In: Proceedings of IJCAI 1999, Sweden, pp. 478–485. Morgan Kaufmann, San Francisco (1999)
6. Brafman, R., Tennenholtz, M.: Learning to Cooperate Efficiently: A Model-Based Approach. Journal of AI Research 19, 11–23 (2003)
7. Buffet, O., Dutech, A., Charpillet, F.: Shaping Multi-Agent Systems with Gradient Reinforcement Learning. Autonomous Agent and Multi-Agent System Journal 15(2), 197–220 (2007)
8. Ernst, D., Geurts, P., Wehenkel, L.: Tree-Based Batch Mode Reinforcement Learning. Journal of Machine Learning Research (6), 504–556 (2005)
9. Gabel, T., Riedmiller, M.: Adaptive Reactive Job-Shop Scheduling with Learning Agents. International Journal of Information Technology and Intelligent Computing 2(4) (2007)
10. Gabel, T., Riedmiller, M.: Reinforcement Learning for DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies. In: Proceedings of AAMAS 2008, Estoril, Portugal, pp. 1333–1336. IFAAMAS (2008)
11. Goldman, C., Zilberstein, S.: Optimizing Information Exchange in Cooperative Multi-Agent Systems. In: Proceedings of AAMAS 2003, Melbourne, Australia, pp. 137–144. ACM Press, New York (2003)
12. Lauer, M., Riedmiller, M.: An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In: Proceedings of ICML 2000, Stanford, USA, pp. 535–542. AAAI Press, Menlo Park (2000)
13. Pinedo, M.: Scheduling. Theory, Algorithms, and Systems. Prentice Hall, Englewood Cliffs (2002)
14. Riedmiller, M.: Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS, vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
15. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: Ruspini, H. (ed.) Proceedings of ICNN, San Francisco, USA, pp. 586–591 (1993)
16. Szer, D., Charpillet, F.: Coordination through Mutual Notification in Cooperative Multiagent RL. In: Proceedings of AAMAS 2004, pp. 1254–1255. IEEE Computer Society, Los Alamitos (2005)
17. Verbeeck, K., Nowe, A., Tuyls, K.: Coordinated Exploration in Multi-Agent Reinforcement Learning: An Application to Load-Balancing. In: Proceedings of AAMAS 2005, Utrecht, The Netherlands, pp. 1105–1106. ACM Press, New York (2005)
18. Watkins, C., Dayan, P.: Q-Learning. Machine Learning 8, 279–292 (1992)

Bayesian Reward Filtering

Matthieu Geist[1,2], Olivier Pietquin[1], and Gabriel Fricout[2]

[1] Supélec
IMS Research Group, Metz, France
{matthieu.geist,olivier.pietquin}@supelec.fr
[2] ArcelorMittal Research,
MCE Department, Maizières-lès-Metz, France
gabriel.fricout@arcelormittal.com

Abstract. A wide variety of function approximation schemes have been applied to reinforcement learning. However, Bayesian filtering approaches, which have been shown efficient in other fields such as neural network training, have been little studied. We propose a general Bayesian filtering framework for reinforcement learning, as well as a specific implementation based on sigma point Kalman filtering and kernel machines. This allows us to derive an efficient off-policy model-free approximate temporal differences algorithm which will be demonstrated on two simple benchmarks.

Keywords: Reinforcement Learning, Function Approximation, Bayesian Filtering.

1 Introduction

Reinforcement learning [1] is a general paradigm in which an agent learns to control a dynamic system only through interactions. A feedback signal is provided to it as a reward information, which is a hint on the quality of the control. Markov Decision Processes (MDP) are a common framework to solve this problem. A MDP is fully described by a tuple $\{S, A, T, R\}$ where S is the state space that can be explored, A is the action set that can be chosen by the agent, T is a family of transition probabilities between states conditioned by the actions and R is a set of expected rewards associated to transitions. This is further explained in section 2.1. In this framework, at each time step k, the system adopts a state s_k. According to this, the agent can chose an action a_k included in a subset of A. This action leads to a transition to state s_{k+1} and to the obtention of a reward r_k, the agent's objective being to maximize the future expected cumulative rewards. Actions can be of two kinds: exploitative or explorative. Exploitative actions are optimal according to the agent's knowledge about the system. Explorative actions aim at increasing the agent's knowledge about the system. This is known as the exploitation vs exploration dilemma. In this paper the knowledge of the environment will be modelled as a Q-function which maps state action pairs to the expected cumulative rewards when following a given associated policy after the first transition. The proposed approach is model-free, no model of transitions and reward distributions is learned or known.

Solutions exist for the reinforcement learning problem with discrete state and action spaces. However they generally do not scale very well and cannot handle continuous state and/or action spaces. A wide variety of function approximation schemes have thus been applied to reinforcement learning (see [1] as a starting point). This is known as the generalization problem, and we propose to handle it with a Bayesian filtering approach.

The objective of Bayesian filtering [2] is to infer a hidden state sequentially from observations. The state evolution is driven by a possibly nonlinear mapping combined with a random process noise. Observations are linked to hidden states through another possibly nonlinear mapping combined with a random observation noise. Thus hidden states and observations are random variables. Under classical assumptions, the analytical solution to this problem is given by the recursive Bayes equations. As they are usually intractable, specific solutions have been proposed. This is further developed in section 2.2.

In this paper are proposed the premises of a Bayesian filtering framework for reinforcement learning. The general idea is to parameterize the Q-function, and to consider the associated parameter vector as the hidden state of a Bayesian filter. The associated observation equation links the reward to the parameters through the Bellman equation [3]. We propose a specific implementation of this general Bayesian reward filter: the Q-function is expressed as a weighted sum of Gaussian kernels, the prior on these kernels is chosen with a dictionary method [4], and the parameter vector evolution is computed with a sigma point Kalman filter [5].

The idea to use Bayesian filtering for reinforcement learning is not novel, but it has been surprisingly little studied. In [6] a modification of the linear quadratic Gaussian Kalman filter model is proposed, which allows the on-line estimation of optimal control (which is off-line for the classical one). In [4] Gaussian processes are used for reinforcement learning. This method can be understood as an extension of the Kalman filter to an infinite dimensional hidden state (the Gaussian process), but it can only handle SARSA-like update rules (because of the necessary linearity of the observation equation), contrarily to the proposed contribution, which can be seen as a nonlinear extension of the parametric case developed in [4]. In [7] a Kalman filter bank is used to find the parameters of a piecewise linear approximation of the value function.

The rest of this paper is organized as follows. First the necessary background is briefly presented. Then the proposed general framework and a specific implementation are explained. It is followed by the first results on two benchmarks: the wet-chicken and the mountain car problems. Eventually we conclude and sketch our future works.

2 Background

In this section will be presented the classical reinforcement learning formalism and the Q-learning algorithm, as well as the general Bayesian filtering paradigm and more specific solutions. Through the rest of the paper, a variable x will denote a column vector or a scalar, which should be clear from the context.

2.1 Reinforcement Learning

A Markov Decision Process (MDP) consists of a state space S, an action space A, a Markovian transition probability $p : S \times A \rightarrow \mathcal{P}(S)$ and a bounded reward function $r : S \times A \times S \rightarrow \mathbb{R}$. A policy is a mapping from state to action space: $\pi : S \rightarrow A$. At time step k, the system is in a state s_k, the agent chooses an action $a_k = \pi(s_k)$, and the system is then driven in a state s_{k+1} following the conditional probability distribution $p(.|s_k, a_k)$. The agent receives the associated reward $r_k = r(s_k, a_k, s_{k+1})$. Its goal is to find the policy which maximizes the expected cumulative rewards, that is the quantity $E_\pi[\sum_{k\in\mathbb{N}} \gamma^k r(S_k, A_k, S_{k+1})|S_0 = s_0]$ for every possible starting state s_0, the expectation being over the state transitions taken upon executing π, where $\gamma \in [0, 1[$ is a discount factor.

A classical approach to solve this optimization problem is to introduce the Q-function defined as:

$$Q^\pi(s,a) = \int_S p(z|s,a)\Big(r(s,a,z) + \gamma Q^\pi(z,\pi(z))\Big)dz$$

It is the expected cumulative rewards by taking action a in state s and then following the policy π. The optimality criterion is to find the policy π^* (and associated Q^*) such that for every state s and for every policy π, $\max_{a\in A} Q^*(s,a) \geq \max_{a\in A} Q^\pi(s,a)$. The optimal Q-function Q^* satisfies Bellman's equation:

$$Q^*(s,a) = \int_S p(z|s,a)\left(r(s,a,z) + \gamma \max_{b\in A} Q^*(z,b)\right)dz$$

In the case of discrete and finite action and state spaces, the Q-learning algorithm provides a solution to this problem. Its principle is to update a tabular approximation of the optimal Q-function after each transition (s, a, r, s'):

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left(r + \gamma \max_{b\in A} \hat{Q}(s',b) - \hat{Q}(s,a)\right)$$

where α is a learning rate. An interesting fact is that the Q-learning is an off-policy algorithm, that is it allows to learn the optimal policy (from the learned optimal Q-function) while following a suboptimal one, given that it is sufficiently explorative. The proposed contribution can be seen as an extension of this algorithm to a Bayesian filtering framework (however with other advantages). See [1] for a comprehensive introduction to reinforcement learning, or [8] for a more formal treatment.

2.2 Bayesian Filtering

The problem of Bayesian filtering is to sequentially infer a hidden state x_k given past observations $y_{1:k} = \{y_1, y_2, \ldots, y_k\}$. It can be expressed in its state-space formulation:

$$\begin{aligned} x_{k+1} &= f_k(x_k, v_k) \\ y_k &= g_k(x_k, n_k). \end{aligned}$$

The state evolution is driven by the mapping f_k and the process noise v_k. The observation y_k is a function of the state x_k, corrupted by an observation noise n_k. Some prior knowledge about the system evolution is necessary (such as the possibly nonlinear and non stationary mappings). The general principle is to predict the new state x_k given the previous observations $y_{1:k-1}$, and to correct it given the new observation y_k, according to prediction and correction equations:

$$p(X_k|Y_{1:k-1}) = \int_{\mathcal{X}} p(X_k|X_{k-1})p(X_{k-1}|Y_{1:k-1})dX_{k-1} \text{ (prediction)},$$
$$p(X_k|Y_{1:k}) = \frac{p(Y_k|X_k)p(X_k|Y_{1:k-1})}{\int_{\mathcal{X}} p(Y_k|X_k)p(X_k|Y_{1:k-1})dX_k} \text{ (correction)}.$$

These equations are generally intractable. If the mappings are linear and if the noises n_k and v_k are Gaussian, the optimal solution is given by the Kalman filter: quantities of interest are random variables, and inference (that is prediction of these quantities and correction of them given a new observation) is done online by propagating sufficient statistics through linear transformations. If the mappings are nonlinear (but the noises are still Gaussian), a first solution is to linearize them around the state: it is the principle of the Extended Kalman Filter (EKF), and sufficient statistics are still propagated through linear transformations. Another approach is the Sigma Point Kalman Filter (SPKF) framework [5]. The basic idea is that it is easier to approximate a probability distribution than an arbitrary nonlinear function. A set of so-called sigma points are deterministically computed from the hidden state sufficient statistics. These points are representative of the distribution of interest. The nonlinear mappings of these points are then computed and used to compute sufficient statistics of interest for prediction and correction equations. Algorithm 1 sketches a SPKF update in the case of additive noise, based on the state-space formulation, and using the standard Kalman notations: $x_{k|k-1}$ denotes a prediction, $x_{k|k}$ an estimate (or a correction), $P_{x,y}$ a covariance matrix, $\bar{n}_k$ a mean and k is the discrete time index. The reader can refer to [5] for details. This filter will be used in the specific implementation of the proposed framework. A last method is the particle filter (or sequential Monte Carlo). It is a numerical approach designed for nonlinear mappings and nongaussian noises. See [2] for a survey on Bayesian filtering.

3 The General Framework

Formulation of value function estimation as a Bayesian filtering problem has been already proposed by Engel [4], however with a linear observation model: it allows nonparametric representation of the value function (Gaussian processes) but is mainly dedicated to the evaluation of the followed policy value (on-policy aspect). Our contribution can be seen as a nonlinear extension of Engel's work, however with a parametric representation constraint.

Algorithm 1. SPKF Update

```
**Inputs:** $\bar{x}_{k-1|k-1}$, $P_{k-1|k-1}$
**Outputs:** $\bar{x}_{k|k}$, $P_{k|k}$

***Sigma points computation:***
Compute deterministically the sigma point set $X_{k-1|k-1}$ from $\bar{x}_{k-1|k-1}$ and $P_{k-1|k-1}$;

***Prediction step:***
Compute $X_{k|k-1} = f_k(X_{k-1|k-1}, \bar{v}_k)$;
Compute $\bar{x}_{k|k-1}$ and $P_{k|k-1}$ from $X_{k|k-1}$ and the process noise covariance;

***Correction step:***
Observe $y_k$;
$Y_{k|k-1} = g_k(X_{k|k-1}, \bar{n}_k)$;
Compute $\bar{y}_{k|k-1}$, $P_{y_{k|k-1}}$ and $P_{x_{k|k-1}, y_{k|k-1}}$ from $X_{k|k-1}$, $Y_{k|k-1}$ and the observation noise covariance;
$K_k = P_{x_{k|k-1}, y_{k|k-1}} P^{-1}_{y_{k|k-1}}$; {*Kalman gain*}
$\bar{x}_{k|k} = \bar{x}_{k|k-1} + K_k(y_k - \bar{y}_{k|k-1})$;
$P_{x_{k|k}} = P_{x_{k|k-1}} - K_k P_{y_{k|k-1}} K_k^T$;
```

The Bellman equation can be written as:

$$\begin{aligned} Q^*(s,a) &= r(s,a,s') + \gamma \max_{b \in A} Q^*(s',b) - n_{s,a}(s') \\ \text{with } n_{s,a}(s') &= \int_S p(z|s,a) \Big\{ r(s,a,s') - r(s,a,z) \\ &\quad + \gamma \left(\max_{b \in A} Q^*(s',b) - \max_{b \in A} Q^*(z,b) \right) \Big\} dz. \end{aligned}$$

Being the nonlinear mapping of the random variable $(S'|S = s, A = a)$ of law $p(.|s,a)$, $n_{s,a}$ is indeed a random variable. It can be easily shown that this random variable is centered with finite variance, that is, $r_{\max}$ being the bound on the reward function:

$$\int_S n_{s,a}(z) p(z|s,a) dz = 0 \quad \text{and} \quad \int_S n^2_{s,a}(z) p(z|s,a) dz \leq \left(\frac{r_{\max}}{1-\gamma} \right)^2$$

For an observed transition (s,a,r,s'), the Bellman equation can be rewritten:

$$r(s,a,s') = Q^*(s,a) - \gamma \max_{b \in A} Q^*(s',b) + n_{s,a}(s')$$

This form is very "Q-learning like" and is of primary importance for the proposed framework.

Suppose that the Q-function is parameterized (either linearly or nonlinearly) by a vector θ. The aim is to find a good approximation $\hat{Q}_\theta$ of the optimal Q-function Q^* by observing transitions (s,a,r,s'). This reward regression problem is cast into a state-space representation. For an observed transition (s_k, a_k, r_k, s'_k), it is written as:

$$\begin{aligned} \theta_{k+1} &= \theta_k + v_k \\ r_k &= \hat{Q}_{\theta_k}(s_k, a_k) - \gamma \max_{a \in A} \hat{Q}_{\theta_k}(s'_k, a) + n_k. \end{aligned}$$

Here v_k is an artificial process noise and n_k a centered observation noise including all the stochasticity of the MDP. The framework is thus posed, but is far from being solved. The observation equation is nonlinear (because of the max operator), that's why classical methods such as the Kalman filter cannot be used. Formally, the process noise is null, nevertheless introducing an artificial noise can improve the stability and convergence performances of the filter. A (possibly adaptive) observation noise has to be chosen also, as well as the parameterization for the Q-function. Last but not least, as for each Bayesian approach, a prior on parameters has to be set. A specific implementation of this Bayesian filtering framework is proposed in the next section.

4 Practical Solution

A kernel parameterization is chosen for the Q-function, because of its expressiveness given by the Mercer theorem [9]. A kernel is a continuous, symmetric and positive definite function. Each kernel is a dot product in a (generally higher) dimensional space. More precisely, for each kernel K, there exists a mapping φ from the working space $\mathcal{X}$ to a so-called feature space $\mathcal{F}$ such that $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. This fact is important for the initialisation of the proposed algorithm.

More precisely Gaussian kernels are chosen, and their mean and deviation are considered as parameters:

$$\begin{gathered} \hat{Q}_\theta(s, a) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i) \\ \text{with } K_{\sigma_i^x}(x, x_i) = \exp\left(-(x - x_i)^T (\Sigma_i^x)^{-1} (x - x_i)\right), \\ \text{where } x = s, a,\ \Sigma_i^x = \operatorname{diag}(\sigma_i^x)^2, \\ \text{and } \theta = [(\alpha_i)_{i=1}^p, (s_i^T)_{i=1}^p, (a_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p, ((\sigma_i^a)^T)_{i=1}^p]^T \end{gathered}$$

The operator diag applied on a column vector gives a diagonal square matrix. Note that $K_{(\sigma_i^s, \sigma_i^a)}([s^T, a^T]^T, [s_i^T, a_i^T]^T) = K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i)$ is a product of Gaussian kernels, thus it is a (still Gaussian) kernel.

4.1 Dictionary Computation

A first problem is to choose the number p of kernel functions and the associated prior. A variety of methods can be contemplated, the simplest one being to choose equally spaced kernel fonctions. However the method described below rests on the mathematical signification of kernels and basic algebra, it is thus well motivated. Moreover, it automatically selects the number of basis functions.

To choose the prior kernel functions, a prior is first put on the Gaussian widths $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$. Then a dictionary method proposed by Engel [4] is used to determine the number of kernels and their prior centers. Consider a kernel K, the associated mapping φ and a set of points $X = \{x_1, x_2, \dots\}$. The aim of the dictionary method is to compute a set of p points $\mathcal{D} = \{\tilde{x}_1, \dots, \tilde{x}_p\}$ such that $\varphi(\mathcal{D})$ is an approximate basis of $\varphi(X)$.

This procedure is iterative. Suppose that samples $x_1, x_2, \dots$ are sequentially observed. At time k, a dictionary $\mathcal{D}_{k-1} = (\tilde{x}_j)_{j=1}^{m_{k-1}} \subset (x_j)_{j=1}^{k-1}$ is available where by construction feature vectors $\varphi(\tilde{x}_j)$ are approximately linearly independent in $\mathcal{F}$. Sample x_k is then observed, and is added if $\varphi(x_k)$ is linearly independent on $\mathcal{D}_{k-1}$. To test this, weights $w = (w_1, \dots, w_{m_{k-1}})^T$ have to be computed so as to satisfy

$$\delta_k = \min_w \left\| \sum_{j=1}^{m_{k-1}} w_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2$$

A predefined threshold ν determining the quality of the approximation (and consequently the sparsity of the dictionary) is used. If $\delta_k > \nu$, $x_k = \tilde{x}_{m_k}$ is added to the dictionary, otherwise $\varphi(x_k)$ can be written as:

$$\varphi(x_k) = \sum_{i=1}^{m_{k-1}} w_i \varphi(\tilde{x}_i) + \varphi_{res} \text{ with } \|\varphi_{res}\|^2 \leq \nu$$

Defining the $m_{k-1} \times m_{k-1}$ matrix $\tilde{K}_{k-1}$ and the $m_{k-1} \times 1$ vector $\tilde{k}_{k-1}(x)$ as

$$\left(\tilde{K}_{k-1}\right)_{i,j} = K(\tilde{x}_i, \tilde{x}_j) \text{ and } \left(\tilde{k}_{k-1}(x)\right)_i = K(x, \tilde{x}_i)$$

and by using the bilinearity of the dot product and the kernel trick, δ_k can be written as:

$$\delta_k = \min_w \left\{ w^T \tilde{K}_{k-1} w - 2 w^T \tilde{k}_{k-1}^T(x_k) + K(x_k, x_k) \right\}$$

whereof solution is $\delta_k = K(x_k, x_k) - \tilde{k}_{k-1}(x_k) w_k$ with $w_k = \tilde{K}_{k-1}^{-1} \tilde{k}_{k-1}(x_k)$. Moreover there exists a computationally efficient algorithm using the partitioned matrix inversion formula to construct this dictionary. See [4] for details. Practically it is supposed that S and A are compact sets and that their bounds are known. Then the corresponding dictionary is computed in a preprocessing step from N samples uniformly sampled from $S \times A$.

4.2 Gaussian Prior

Recall that as for any Bayesian approach, a prior parameter distribution has to be chosen. We state that $\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0})$ where

$$\bar{\theta}_0 = [\alpha_0, \dots, \mathcal{D}_s, \mathcal{D}_a, (\sigma_0^s)^T, \dots, (\sigma_0^a)^T, \dots]^T$$
$$\Sigma_{\theta_0} = \operatorname{diag}(\sigma_{\alpha_0}^2, \dots, \sigma_{\mu_0^s}^2, \dots, \sigma_{\mu_0^a}^2, \dots, \sigma_{\sigma_0^s}^2, \dots, \sigma_{\sigma_0^a}^2, \dots)$$

In these expressions, α_0 is the prior mean on kernel weights, $\mathcal{D} = \mathcal{D}_s \times \mathcal{D}_a$ is the set of prior means on kernel centers computed in a preprocessing step with the dictionary from the prior means on kernel deviations $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$, and $\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0^x}$, $\sigma^2_{\sigma_0^x}$ are respectively the prior variances on kernel weights, centers and deviations, for $x = s, a$. All these parameters (except the dictionary) have to be initialized beforehand, using the problem's prior knowledge. Let $q = (2(n_a + n_s) + 1)\, p$, with n_s (resp. n_a) being the dimension of the state space (resp. the action space). Note that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q \times q}$. A prior on noises also has to be put: more precisely, $v_0 \sim \mathcal{N}(0, R_{v_0})$ and $n_0 \sim \mathcal{N}(0, R_{n_0})$, where $R_{n_0} = \sigma^2_{n_0}$.

4.3 Parameters Update

Once the parameters are initialized, the parameter vector has still to be updated as new observations (s_k, a_k, r_k, s'_k) are available. A Square-Root Central Difference Kalman Filter (SR-CDKF) is used, which is a specific implementation of the SPKF sketched before. See [5] for further information. The last problem is to choose the artificial process noise. Formally, since the target function is stationary, there is no process noise. However, following [5], an artificial process noise is introduced.

Its covariance is set to a fraction of the parameter covariance, that is

$$R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$$

where $\lambda \in]0, 1]$ $(1 - \lambda \ll 1)$ is a forgetting factor similar to the one from the recursive least-squares (RLS) algorithm. In this paper, the observation noise is constant, that is $R_{n_k} = R_{n_{k-1}}$. The proposed Bayesian reward filtering algorithm is summarized in Algorithm 2.

4.4 Maximum over Action Space

Notice that a technical difficulty is to compute the maximum over the actions for the parameterized Q-function. This computation is necessary for the filter update. A first solution is to sample the action space and to compute the maximum over the obtained samples. However this is especially computationally inefficient. The used method is closed to one proposed in [10].

The maximum over action kernel centers is computed: $\mu^a = \operatorname{argmax}_{a_i} \hat{Q}_\theta(s, a_i)$. It serves then as the initialization for the Newton-Raphson method used to find the maximum :

$$a_m \leftarrow a_m - \left((\nabla_a \nabla_a^T)\hat{Q}_\theta(s, a)\right)^{-1}_{a=a_m} \nabla_a \hat{Q}_\theta(s, a)\Big|_{a=a_m}$$

If the Hessian matrix is singular, a gradient ascent/fixed point scheme is used:

$$a_m \leftarrow a_m + \nabla_a \hat{Q}_\theta(s, a)\Big|_{a=a_m}$$

The obtained action a_m is considered as the action which maximizes the parameterized Q-function.

Algorithm 2. A Bayesian reward filtering algorithm

```
inputs: $\nu$, $N$, $\alpha_0$, $\sigma_0$, $\sigma_{\alpha_0}$, $\sigma_{\mu_0^x}$, $\sigma_{\sigma_0^x}$, $\sigma_{v_0}$, $\sigma_{n_0}$
outputs: $\bar{\theta}$, $\Sigma_\theta$

Compute Engel's dictionary:
$\forall i \in \{1 \dots N\}$, $[s_i^T, a_i^T]^T \sim \mathcal{U}_{\mathcal{S}\times\mathcal{A}}$;
Set $X = \{[s_1^T, a_1^T]^T, \dots, [s_N^T, a_N^T]^T\}$;
$\mathcal{D}$ =Compute-Dictionary($X$,$\nu$,$\sigma_0$)

Initialization:
Initialize $\bar{\theta}_0$, $\Sigma_{\theta_0}$, $R_{n_0}$, $R_{v_0}$;

for $k = 1, 2, \dots$ do
    Observe $t_k = (s_k, a_k, r_k, s'_k)$;

    SR-CDKF update:
    $[\bar{\theta}_k, \Sigma_{\theta_k}]$ = SR-CDKF-Update($\bar{\theta}_{k-1}$,$\Sigma_{\theta_{k-1}}$,$t_k$,$R_{v_{k-1}}$,$R_{n_{k-1}}$);

    Artificial process noise update:
    $R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$;
end for
```

5 Preliminary Results

The proposed approach is demonstrated on two problems. First, the "wet-chicken" task is a continuous state and action space and stochastic problem. Second, the "mountain car" problem is a continuous state, discrete action and deterministic problem. The latter will require an hybrid parametrization.

Notice that this section focuses on the regression part of the proposed method, that is the ability to learn an optimal Q-function from random transitions. Thereby the algorithm does not learn from trajectories here, and thus notably avoid the dilemma between exploration and exploitation, which is left for future works. Let's discuss the choice of parameters.

5.1 Choice of Parameters

For both tasks the reinforcement learning discount factor was set to $\gamma = 0.9$. The dictionary's sparsity factor was set to $\nu = 0.9$. Similarly to the recursive least-squares algorithm, the adaptive process noise covariance was set to a high value, such that $\lambda^{-1} - 1 \simeq 10^{-6}$. The dictionary is computed from $N = 10^4$ samples uniformly sampled from $S \times A$. Note that even if this is done in a preprocessing step, the learning part of the algorithm is nonetheless online.

The initial choice of kernel deviations is problem dependant. However a practical good choice seems to take a fraction of the quantity $x(j)_{\max} - x(j)_{\min}$ for the kernel deviation associated to $x(j)$, the j^{th} component of the column vector x, $x(j)_{\max}$ and $x(j)_{\min}$ being the bounds on the values taken by the variable $x(j)$. We supposed the prior kernel weights to be centered, and we set the associated standard deviation to a little fraction of the theoretical bound on Q-function, that is $\frac{r_{\max}}{1-\gamma}$. Because of geometry of Gaussian distributions, we suppose that centers provided by the dictionary are approximately uniformly distributed, and we set the prior deviation of the j^{th} component of the vector x to a little fraction of $(x(j)_{\max} - x(j)_{\min})p^{-\frac{1}{ns+na}}$, with the convention that for discrete spaces $n = 0$. Finally we set up the deviation of the prior kernel deviations to a little fraction of them. Otherwise speaking, we set $\sigma_{\sigma_0^{x(j)}}$ to a little fraction of $\sigma_0^{x(j)}$ for the j^{th} component of x.

To sum up, the Gaussian prior on parameterization is chosen such that:

$$\sigma_0^{x(j)} \propto x(j)_{\max} - x(j)_{\min}$$
$$\mu_{\alpha_0} = 0 \text{ and } \sigma_{\alpha_0} \propto \frac{r_{\max}}{1-\gamma}$$
$$\sigma_{\mu_0^{x(j)}} \propto \frac{x(j)_{\max} - x(j)_{\min}}{\sqrt[(n_s+n_a)]{p}}$$
$$\sigma_{\sigma_0^{x(j)}} \propto \sigma_0^{x(j)}$$

5.2 Wet-Chicken

In the wet-chicken problem (inspired by [11]), a canoeist has to paddle on a river until reaching a waterfall. It restarts if it falls down. Rewards increase linearly with the proximity of the waterfall, and drop off for falling. Turbulences make the transition probabilistic. More formally, the state space is $S = [0, 10]$ (10 being the waterfall position), the action space $A = [-1, 1]$ (continuously from full backward padding to full forward padding), the transition is $s' = s + a + c$ with $c \sim \mathcal{N}(0, \sigma_c)$, $\sigma_c = 0.3$ and the associated reward is equal to $r = \frac{s'}{10}$. If $s' \geq 10$ the canoeist falls, the associated reward is $r = -1$ and the episode ends.

To test the proposed framework, random transitions are uniformly sampled and used to feed the filter: at each time step k, a state s_k and an action a_k are uniformly sampled from $S \times A$, and used to generate a (random) transition to s'_k, with associated reward r_k, and the transition (s_k, a_k, s'_k, r_k) is the input of the algorithm. The results are shown on Fig. 1. For each run of the algorithm and every 250 samples, the expected cumulative rewards for the current policy has been computed as an average of cumulative rewards over 1000 episodes which were randomly (uniform distribution) initiated (thus the average is done over starting states and stochasticity of transitions). Notice that the lifetime of the agent (the duration of an episode) was bounded to 100 interactions with its environment. We then computed a two dimensional histogram of those averaged cumulative rewards over 100 different runs of the algorithm. In other words, we show the distribution of cumulative rewards over different runs of the algorithm as a function of the number of observed transitions. The bar on the right shows the percentages associated to the histogram.

The optimal policy has an averaged cumulative rewards of 6 (computed with value iteration over a very finely sampled state-action space). One can see on Fig. 1 that the proposed algorithm can learn near optimal policies. After 1000 samples some of policies can achieve a score of 5 (84% of the optimal policy),

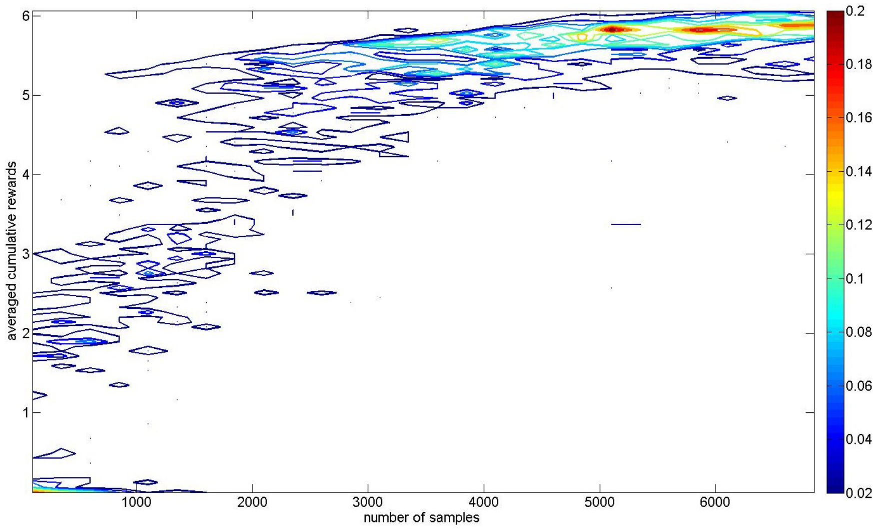


Fig. 1. Two-dimensional histogram of averaged cumulative rewards for the wet-chicken problem (100 runs)

which is achieved by a majority of the policies after 3000 samples. After 7000, very close to optimal policies were found in almost all runs of the algorithm (the mode of the associated distribution is at 5.85, that is 98% of the optimal policy). To represent the approximated Q-function, 7.7 ± 0.7 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

Two remarks of interest have to be made on this benchmark. First, the observation noise is input-dependant, as it models the stochasticity of the MDP. Recall that here we have chosen a constant observation noise. Secondly, the noise can be far to Gaussianity. For example, in the proximity of the waterfall it is bimodal because of the shift of reward. Recall that the proposed filter assumes Gaussianity of noises. Thus we can conclude that the proposed approach is quite robust, and that it achieves good performance considering that the observations were totally random (off-policy aspect).

5.3 Mountain Car

The second problem is the mountain-car task. A underpowered car has to go up a steep mountain road. The state is 2-dimensional (position and velocity) and continuous, and there are 3 actions (forward, backward and zero throttle). The problem full description is given in [1]. A null reward is given at each time step, and $r = 1$ is given when the agent reaches the goal.

A first problem is to find a parameterization for this task. The proposed one is adapted for continuous problems, not hybrid ones. But this approach can be easily extended to continuous states and discrete actions tasks. A simple solution consists in having a parameterization for each discrete action, that is a parametrization of the form $\theta = [\theta^{a_1}, \theta^{a_2}, \theta^{a_3}]$ and an associated Q-function $Q_\theta(s,a) = Q_{\theta^a}(s)$. But one can notice that for a fixed state and different actions the Q-values will be very close. In other words $Q^*(s,a_1)$, $Q^*(s,a_2)$ and $Q^*(s,a_3)$ will have similar shapes, as functions over the state space. Thus consider that the weights will be specific to each action, but the kernel centers and deviations will be shared over actions. More formally the parameter vector is

$$\theta = [(\alpha_i^{a_1})_{i=1}^p, (\alpha_i^{a_2})_{i=1}^p, (\alpha_i^{a_3})_{i=1}^p, (s_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p]^T$$

the notation being the same as in the previous sections.

As for the wet-chicken problem, the filter has been fed with random transitions. Results are shown on Fig. 2, which is a two-dimensional histogram similar to the previous one. The slight difference is that the performance measure is now the "cost-to-go" (the number of steps needed to reach the goal). It can be linked directly to the averaged cumulative rewards, however we think it is more meaningful. For each run of the algorithm and every 250 samples, the expected cost-to-go for the current policy has been computed as an average of 1000 episodes which were randomly initiated (average is only done over starting states here, as transitions are deterministic). The lifetime of the agent was bounded to 1000 interactions with its environment. The histogram is computed over 100 runs.

The optimal policy has an averaged cost-to-go of 55 (computed with value iteration over a very finely sampled state space). One can see on Fig. 2 that the proposed algorithm can find near optimal policies. After 1500 samples most of policies achieve a cost-to-go smaller than 120. After 6000 samples, policies very

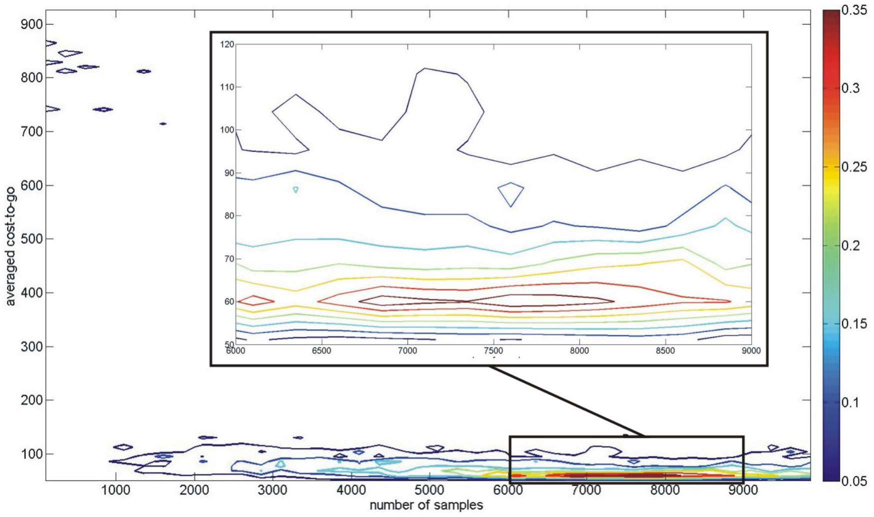


Fig. 2. Two-dimensional histogram of the averaged cost-to-go for the mountain-car problem (100 runs)

close to the optimal one were found in almost all runs of the algorithm (the mode of the associated distribution is at 60). To represent the approximated Q-function, 7.5 ± 0.8 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

This problem is not stochastic, but informative rewards are very sparse (which can cause the Kalman gain to converge too quickly), transitions are nonlinear and rewards are binary. Despite this, the proposed filter exhibits good convergence. Once again we can conclude that the proposed approach achieves good results considering the task at hand.

5.4 Comparison to Other Methods

For now the proposed algorithm treats the different control tasks as regression problems (learning from random transitions), thus it is ill comparable to state-of-the-art reinforcement learning algorithms which learns from trajectories. Nevertheless we argue that the quality of learned policy is comparable to state-of-the art methods. Measuring this quality depends on the problem settings and on the measure of performance, however the Bayesian reward filter finds very close to optimal policies. See [11] for example.

In most approaches, the system is controlled while learning, or for batch methods observed samples come from a suboptimal policy. In the proposed experiments, totally random transitions are observed. However for the mountain-car problem it is often reported that at least a few hundreds of episodes are required to learn a near-optimal policy (see for example [1]), and each episode may contain from a few tens to hundreds steps (this depends on the quality of the current control). In the proposed approach a few thousands of transitions have to be observed in order to obtain a near optimal policy. This is roughly the same order of magnitude for convergence speed.

We have demonstrated the algorithm on two simple benchmarks, notwithstanding it is planned to conduct more extensive comparison to other approaches when a control organ is added to the proposed framework.

6 Conclusion and Future Works

We have introduced a general Bayesian filtering framework for reinforcement learning. By observing rewards (and associated transitions) the filter is able to infer a near-optimal policy (through the parameterized Q-function). A specific implementation, based on sigma point Kalman filtering, on kernel machines and on a sparsification method has been described. It has been tested on two reinforcement learning benchmarks, each one exhibiting specific difficulties for the algorithm. This off-policy Bayesian reward filter has been shown to be efficient on this two continuous tasks.

However, this paper did not demonstrate all the potentialities of the proposed framework. The Bayesian filtering approach allows to derive uncertainty information over estimated Q-function which can be used to handle the exploration-exploitation dilemma, in the spirit of [12] or [13]. This could allow to speed-up and to enhance learning. Moreover, we think that the partial observability problem can be quite naturally embedded in such a Bayesian filtering framework, as the Q-function can be considered as a function over probability densities.

In our future works we aim to treat the two aforementioned problems, our main goal being to handle what we consider to be the three major reinforcement learning problems (generalization, partial observability and exploration-exploitation trade-off) at once. However there are other points of interest. A more efficient adaptive process noise could speed up and improve the convergence's robustness of the filter, and adaptive observation noise could be necessary for some more complex tasks, as it is formally input-dependent. Other parametrization for the Q-function can also be considered. Moreover, there are a few technical issues, as the search of maxima over actions. Last but not least theoretical convergence may be a problem and should be studied.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), 3rd edn. The MIT Press, Cambridge (1998)
2. Chen, Z.: Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. Technical report, Adaptive Systems Lab, McMaster University (2003)
3. Bellman, R.: Dynamic Programming, 6th edn. Dover Publications (1957)
4. Engel, Y.: Algorithms and Representations for Reinforcement Learning. Ph.D thesis, Hebrew University (April 2005)
5. van der Merwe, R.: Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models. Ph.D thesis, OGI School of Science & Engineering, Oregon Health & Science University, Portland, OR, USA (April 2004)
6. Szita, I., Lőrincz, A.: Kalman Filter Control Embedded into the Reinforcement Learning Framework. Neural Comput. 16(3), 491–499 (2004)
7. Phua, C.W., Fitch, R.: Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation. In: ICML 2007 (2007)
8. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 3rd edn. Athena Scientific (1995)
9. Vapnik, V.N.: Statisical Learning Theory. John Wiley & Sons, Inc., Chichester (1998)
10. Carreira-Perpinan, M.A.: Mode-Finding for Mixtures of Gaussian Distributions. IEEE Transactions on Pattern Analalysis and Machine Intelligence 22(11), 1318–1323 (2000)
11. Schneegass, D., Udluft, S., Martinetz, T.: Kernel Rewards Regression: an Information Efficient Batch Policy Iteration Approach. In: AIA 2006: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications, Anaheim, CA, USA, pp. 428–433. ACTA Press (2006)
12. Dearden, R., Friedman, N., Russell, S.J.: Bayesian Q-learning. In: Fifteenth National Conference on Artificial Intelligence, pp. 761–768 (1998)
13. Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC Model-Free Reinforcement Learning. In: 23rd International Conference on Machine Learning (ICML 2006), Pittsburgh, PA, USA, pp. 881–888 (2006)

Basis Expansion in Natural Actor Critic Methods

Sertan Girgin[1] and Philippe Preux[1,2]

[1] Team-Project SequeL, INRIA Lille Nord-Europe
[2] LIFL (UMR CNRS), Université de Lille
{sertan.girgin,philippe.preux}@inria.fr

Abstract. In reinforcement learning, the aim of the agent is to find a policy that maximizes its expected return. Policy gradient methods try to accomplish this goal by directly approximating the policy using a parametric function approximator; the expected return of the current policy is estimated and its parameters are updated by steepest ascent in the direction of the gradient of the expected return with respect to the policy parameters. In general, the policy is defined in terms of a set of basis functions that capture important features of the problem. Since the quality of the resulting policies directly depend on the set of basis functions, and defining them gets harder as the complexity of the problem increases, it is important to be able to find them automatically. In this paper, we propose a new approach which uses cascade-correlation learning architecture for automatically constructing a set of basis functions within the context of Natural Actor-Critic (NAC) algorithms. Such basis functions allow more complex policies be represented, and consequently improve the performance of the resulting policies. We also present the effectiveness of the method empirically.

1 Introduction

Reinforcement learning (RL) is the problem faced by an agent that is situated in an environment and must learn a particular behavior through repeated trial-and-error interactions with it [1]; at each time step, the agent observes the state of the environment, chooses its action based on these observations and in return receives some kind of "reward", in other words a *reinforcement signal*, from the environment as feedback. The aim of the agent is to find a policy, a way of choosing actions, that maximizes its overall gain – a function of rewards, such as the (discounted) sum or average over a time period. Policy gradient methods try to accomplish this goal by directly approximating the policy using a parametric function approximator. Instead of estimating the value function and then deriving a greedy policy with respect to the value function, the expected return of the current policy is estimated and its parameters are updated by steepest ascent in the direction of the gradient of the expected return with respect to the policy parameters. Policy gradient methods benefit from strong convergence properties, and especially with the emergence of methods that utilize the natural gradient estimations that are independent of the coordinate frame chosen for expressing the policy parameters, shown to be effective. In most of these approaches, the policy is represented in terms of a set of problem dependent basis functions. The basis functions map the state(-action) variables to real numbers; each basis function performs a mapping that is different from the others, and aims at capturing the important features of the problem domain and the relations within. Consequently, the quality of the resulting policies directly depend on these functions. However, as the complexity of the problem that is being solved increases, it also becomes harder to find a “good” set of such functions. This brings up the question of what the set of best basis functions is and how we can find them.

In particular, in this paper, we will focus on the Natural Actor Critic (NAC) methods together with a linear combination of basis functions to represent a parameterized policy, and a compatible advantage value function to learn an optimal policy from a given set of experience samples. We seek to provide a possible answer to the question posed above by proposing a method that incorporates a cascade correlation learning architecture into NAC and iteratively adds new basis functions to a set of initial basis functions. In Section 2, we first introduce policy gradient and natural actor critic methods. Sections 3 describes cascade correlation learning architecture followed by the details of the proposed method for basis function expansion. Section 4 presents empirical evaluations on some benchmark problems, and a review of related work can be found in Section 5. Finally, Section 6 concludes.

2 Policy Gradient and Natural Actor Critic Methods

A *Markov decision process* (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}(s, a, s')$ is the transition function which denotes the probability of making a transition from state s to state s' by taking action a, $\mathcal{R}(s, a)$ is the expected reward function when taking action a in state s. A *policy* is a probability distribution over actions conditioned on the state; $\pi_\theta(s, a)$ denotes the probability that policy π selects action a at state s. We assume that the policy is parameterized with (a small number) of parameters θ, and is differentiable with respect to these parameters, i.e. that $\partial\pi_\theta(s, a)/\partial\theta$ exists. The state-value function $V^\pi(s)$ is a mapping from states to real numbers, where the value of a state represents the expected return starting from that state, and following policy π; it is given by

$$V^\pi(s) = E_{a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$$

where $\gamma \in [0, 1]$ is the discount factor, r_t is the reward received at time t, and a_t denotes the action drawn from policy π at time t. Similarly, state-action value function $Q^\pi(s, a)$ is defined as the expected return when taking action a at state s, and following policy π thereafter:

$$Q^\pi(s, a) = E_{a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$$

Our aim is to find an optimal policy π^* that maximizes the expected return

$$\rho(\pi) = E\{\sum_{t=0}^{\infty} \gamma^t r_t | s_0, \pi\} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi(s,a)\mathcal{R}(s,a) ds da$$

where $d^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s|s_0, \pi\}$ is the discounted distribution of states under π[1].

In policy gradient methods, an optimal policy is sought by an iterative process; starting from an initial policy, at each iteration the gradient $\nabla_\theta \rho(\pi_{\theta_k})$ of the expected return for the current policy π_θ is estimated, and then the policy parameters θ_k are updated in the direction of the gradient by gradient ascent

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \rho(\pi_{\theta_k}) \tag{1}$$

where α_k is the learning rate. The estimation step is called *policy evaluation*, and the update step is called *policy improvement*; the entire process falls into the general framework of policy iteration [2,3]. Policy gradient methods possess two important properties, (i) small changes in θ results in small changes in the policy and in the distribution of states under that policy, and (ii) if the gradient estimate is unbiased and learning rates are square summable but not summable (i.e. $\sum_{k=0}^{\infty} \alpha_k > 0$ and $\sum_{k=0}^{\infty} \alpha_k^2 = c$) then it is guaranteed to converge to a (locally) optimal policy. The first property brings an important advantage over value-function based methods[2] as small changes in the value estimations may lead to arbitrary changes in the associated policy, hindering convergence.

The critical problem that needs to be solved in the policy evaluation step is to obtain a "good" estimate of the gradient. A simple way to do this is to use the method of finite differences. In this approach, the policy parameters are varied by small perturbations, and then roll-outs are performed for each perturbed policy to generate estimates of the change in the expected return; regression of perturbations onto these changes yields the gradient. The major drawback of this approach is that the amount of perturbations usually differ for each parameter and they must be chosen carefully for a successful estimation. Also, it is sensitive to noise. An alternative approach is to apply likelihood ratio principle, which leads to the policy gradient theorem stating that

$$\frac{\partial \rho}{\partial \theta} = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \frac{\partial \pi(s,a)}{\partial \theta} (Q^{\pi}(s,a) - b^{\pi}(s)) ds da \tag{2}$$

where $b^{\pi}(s)$ is a baseline function [4]. As there are no terms regarding the gradient of the state distribution $d^{\pi}(s)$ with respect to θ, the gradient of the expected

[1] When one is interested in maximizing the average reward rather than the discounted case, we have $\rho(\pi) = \lim_{n\to\infty} \frac{1}{n} E\{r_t|\pi\}$ and $d^{\pi}(s)$ becomes the stationary distribution of the states under π.

[2] In value-function based methods, the focus is on parameterizing and approximating the value function rather than the policy, and the policy is represented implicitly as being greedy with respect to the estimated value function.

return does not depend on changes in the distribution of states, and hence an unbiased estimate can be obtained from states sampled from the state distribution obtained by following π. If one uses the actual returns calculated over sample trajectories to approximate $Q^\pi(s,a)$, this formula reduces to the (episodic) REINFORCE algorithm of Williams [5]. The baseline $b^\pi(s)$ can be an arbitrary function of s; it does not introduce any bias but can minimize the variance of the estimate if chosen accordingly. In [4] and [6], it has been shown that the $Q^\pi(s,a) - b^\pi(s)$ term in Equation 2 can be replaced by a compatible linear function approximation $f^\pi_\omega(s,a)$ of the form

$$f^\pi_\omega(s,a) = \nabla_\theta \log \pi(s,a)^T \omega \tag{3}$$

without affecting the bias of the estimation, giving

$$\begin{aligned}\frac{\partial\rho}{\partial\theta} &= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \frac{\partial\pi(s,a)}{\partial\theta} \nabla_\theta \log \pi(s,a)^T dsda\, \omega &(4)\\ &= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi(s,a)\nabla_\theta \log \pi(s,a) \nabla_\theta \log \pi(s,a)^T dsda\, \omega &(5)\\ &= F(\theta)\omega &(6)\end{aligned}$$

as $\nabla_\theta \pi(s,a) = \pi(s,a)\nabla_\theta \log \pi(s,a)$. $F(\theta)$ is called the all-action matrix. Note that, since

$$\int_{\mathcal{A}} \pi(s,a) f^\pi_\omega(s,a) = 0 \tag{7}$$

$f^\pi_\omega(s,a)$ actually approximates the advantage value function, i.e. in fact $b^\pi(s) = V^\pi(s)$ and we have

$$f^\pi_\omega \approx A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{8}$$

An important result regarding Equation 5 as demonstrated by Peters and Schaal is that $F(\theta)$ corresponds to the Fisher information matrix [7]; this has an interesting and useful consequence as we will see shortly. In [8], Amari showed that for a function $L(\omega)$ defined on a parameter space $S = \{\omega \in \Re^n\}$, if ω is not an orthonormal coordinate system, i.e. S is a Riemannian space, then the steepest ascent direction of $L(\omega)$ in S is not equal to the gradient of $L(\omega)$ with respect to ω, $\nabla_\omega L(\omega)$, but is given by

$$\tilde{\nabla}_\omega L(\omega) = G^{-1}(\omega)\nabla_\omega L(\omega) \tag{9}$$

where $G(\omega)$ denotes the Fisher information matrix. $\tilde{\nabla}_\omega L(\omega)$ is called the *natural gradient*. It is guaranteed that the angle between $\nabla_\omega L(\omega)$ and $\tilde{\nabla}_\omega L(\omega)$ is never larger than ninety degrees and following the natural gradient convergences to a (local) optimum. In most cases, the steepest descent with respect to the natural gradient seems more efficient than normal gradients, especially when the gradients are small and do not point directly toward the optimal solution. Normally, one needs to construct the Fisher information matrix and find its inverse in order to calculate the natural gradient. However, in case of policy gradient with compatible function approximation, combining Equations 1, 5 and 9, Fisher information matrix and its inverse cancel each other, and one ends up with a very simple update rule for the policy parameters:

$$\theta_{k+1} = \theta_k + \alpha_k \omega \tag{10}$$

where the update terms become the weights of $f_\theta^\pi(s,a)$.

In order to take advantage of this simple formulation and convert it into an effective reinforcement learning algorithm, we need to be able to approximate $f_\theta^\pi(s,a)$ from the samples generated from a model or collected from interactions with the environment. Unlike state-value and state-action value functions, it is not possible to learn $f_\theta^\pi(s,a)$ solely using temporal difference like bootstrapping methods, as the value of states, i.e. $V^\pi(s)$, that are required for comparison are subtracted in the advantage function. In order to remedy this situation, one approach might be to have a separate approximation for the value function. Note that, by definition $Q^\pi(s,a) = f_\theta^\pi(s,a) + V^\pi(s)$, and the Bellman equation can be written as

$$f_\theta^\pi(s,a) + V^\pi(s) = \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} P(s,a,s')V^\pi(s')ds \tag{11}$$

Given a set of samples (s_t, a_t, r_t, s_{t+1}), the Natural Actor-Critic (NAC) algorithm proposed by Peters et. al uses a linear function approximation of the value function in the form of

$$V^\pi(s) = \phi(s)^T v$$

with appropriate basis functions $\phi(s)$ to construct a set of simultaneous linear equations

$$\nabla_\theta \log \pi(s_t,a_t)^T \omega + \phi(s_t)^T v \approx r_t(s,a) + \gamma\phi(s_{t+1})^T v \tag{12}$$

and solves Equation 11 using the $LSTD(\lambda)$ policy evaluation algorithm [7,9]. $LSTD(\lambda)$ is shown to converge with probability one for function approximation, hence with appropriate basis functions NAC also succeeds in finding the true natural gradient and converges to a (local) optima in the Riemannian space. In this approach, not only the parametrization of the policy but also the basis functions chosen to represent the value function play an important role, and have a direct effect on the quality of the solution. An alternative approach, which does not require this dependency is the episodic Natural Actor-Critic (eNAC) algorithm [9,7]. Given a trajectory of samples $(s_0, a_0, r_0, s1) \ldots (s_n, a_n, r_n, s_{n+1})$, by summing Equation 12 over the trajectory one obtains:

$$\sum_{t=0}^{n} \gamma^t A^\pi(s_t,a_t) = V^\pi(s_0) + \sum_{t=0}^{n} \gamma^t r_t - \gamma^{n+1} V^\pi(s_{n+1})$$

Ignoring the last term, and assuming a single start state s_0 for all trajectories, from a set of trajectories we can define a set of linear equations

$$\sum_{t=0}^{n} \gamma^t \nabla_\theta \log \pi(s_t,a_t)^T \omega + \rho = \sum_{t=0}^{n} \gamma^t r_t$$

with $|\omega|+1$ unknowns, which can be solved in a least squares sense to find the natural gradient.

Natural policy gradient based methods have been shown to be quite effective and perform better than regular policy gradient methods in various settings under the condition that the learning process is started with a reasonable policy [7,10,11]. Nonetheless, the performance of the resulting policies, at the end, depend on their structure and expression power. In practice, policies are generally represented as parametric linear mixtures of basis functions; consequently, the set of basis functions and the initial weights emerge as the determining factors. As the complexity of the problem increases, it also gets progressively more difficult to come up with a good set of basis functions. Generic approaches, such as regular grids or regular radial basis function networks, which are quite successful in small problems, become impractical due to the exponential growth of the state and action spaces with respect to their dimension. Therefore, given a problem, it is highly desirable to determine a compact set of such basis functions automatically. In the next section, we will first describe a particular class of a function approximator and learning architecture called *cascade-correlation networks*, and then we will present how they can be utilized in order to iteratively construct new basis functions.

3 Cascade Correlation Networks and Basis Function Construction

Cascade correlation is both an architecture and a supervised learning algorithm for artificial neural networks introduced by [12]. It aims to overcome *step-size* and *moving target* problems that negatively affect the performance of backpropagation learning algorithm. Similar to traditional neural networks, the neuron is the most basic unit in cascade correlation networks. However, instead of having a predefined topology with the weights of the fixed connections between neurons getting adjusted, a cascade correlation network starts with a minimal structure consisting only of an input and an output layer, without any hidden

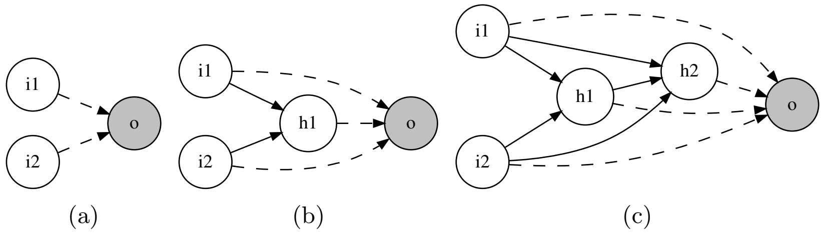


Fig. 1. (a) Initial configuration of a simple cascade-correlation network with two inputs and a single output (in gray). (b) and (c) show the change in the structure of the network as two new hidden nodes are subsequently added. Solid edges indicate input weights that stay fixed after the candidate training phase.

layer. All input neurons are directly connected to the output neurons (Figure 1a). Then, the following steps are taken:

1. All connections leading to output neurons are trained on a sample set and corresponding weights (i.e. only the input weights of output neurons) are determined by using an ordinary learning algorithm until the error of the network no longer decreases. This can be done by applying the regular "delta" rule, or using more efficient methods such as quick-prop or rprop. Note that, only the input weights of output neurons (or equivalently the output weights of input neurons) are being trained, therefore there is no back-propagation.
2. If the accuracy of the network is above a given threshold then the process terminates.
3. Otherwise, a set of *candidate units* is created. These units typically have non-linear activation functions, such as sigmoid or Gaussian. Every candidate unit is connected with all input neurons and with all existing hidden neurons (which is initially empty); the weights of these connections are initialized randomly. At this stage the candidate units are not connected to the output neurons, and therefore are not actually active in the network. Let s denote a training sample. The connections leading to a candidate unit are trained with the goal of maximizing the sum $\mathcal{S}$ over all output units o of the magnitude of the correlation between the candidate units value denoted by v_s, and the residual error observed at output neuron o denoted by $e_{s,o}$. S is defined as

$$S = \sum_o |\sum_s (v_s - v)(e_{s,o} - e_o)|$$

where v and e_o are the values of v_s and $e_{s,o}$ averaged over all samples, respectively. As in step 1, learning takes place with an ordinary learning algorithm by performing gradient ascent with respect to each of the candidate units incoming weights:

$$\frac{\partial S}{\partial w_i} = \sum_{s,o} (e_{s,o} - e_o)\sigma_o f'_s I_{i,s}$$

where σ_o is the sign of the correlation between the candidates value and output o, f'_s is the derivative for sample s of the candidate units activation function with respect to the sum of its inputs, and $I_{i,s}$ is the input the candidate unit received from neuron i for sample s. Note that, since only the input weights of candidate units are being trained there is again no need for back-propagation. Besides, it is also possible to train candidate units in parallel since they are not connected to each other. By training multiple candidate units instead of a single one, different parts of the weight space can be explored simultaneously. This consequently increases the probability of finding neurons that are highly correlated with the residual error. The learning of candidate unit connections stops when the correlation scores no longer improve or after a certain number of passes over the training set. Now, the candidate unit with the maximum correlation is chosen, its incoming weights are frozen (i.e. they are not updated in the subsequent steps) and it

   is added permanently to the network by connecting it to all output neurons (Figure 1b and c). The initial weights of these connections are determined based on the value of correlation of the unit. All other candidate units are discarded.
4. Return back to step 1.

Until the desired accuracy is achieved at step 2, or the number of neurons reaches a given maximum limit, a cascade correlation network completely self-organizes itself and grows as necessary. One can easily observe that, by adding hidden neurons one at a time and freezing their input weights, training of both the input weights of output neurons (step 1) and the input weights of candidate units (step 3) reduce to one step learning problems. Since there is no error to back-propagate to previous layers the moving target problem is effectively eliminated. Also, by training candidate nodes with different activation functions and choosing the best among them, it is possible to build a more compact network that better fits the training data.

One observation here is that, unless any of the neurons has a stochastic activation function, the output of a neuron stays constant for a given sample input. This brings the possibility of storing the output values of neurons which in return reduces the number of calculations in the network and improve the efficiency drastically compared to traditional multi-layer back-propagation networks, especially for large data sets. But more importantly, *each hidden neuron effectively becomes a permanent feature detector*, or to put it another way, a basis function in the network; the successive addition of hidden neurons in a cascaded manner allows, and further, facilitates the creation of more complex feature detectors that helps to reduce the error and better represent the functional dependency between input and output values. We would like to point out that, this entire process does not require any user intervention and is well-matched to our primary goal of determining a set of good basis functions for representing the policy and the corresponding compatible advantage value function in natural actor-critic methods.

As described in Section 2, in Natural Actor-Critic methods, we have a parameterized policy and a compatible approximation to the advantage value function, which is a linear combination of the partial derivatives of the logarithm of the policy with respect to the policy parameters. For the sake of simplicity, in the discussion below we will consider the case in which there is only one control variable; the extension to multiple control variables is trivial and easily follows. We will assume that the stochastic policy with policy parameters θ is defined by a set of basis functions $\boldsymbol{\varphi} = \{\varphi_1(s), \ldots, \varphi_n(s)\}$, each basis function being a function of state, and is in the form of a normal distribution with mean $\boldsymbol{\varphi}^T\theta$ and variance σ^2:

$$\pi_\theta(s, a) = \mathcal{N}(\boldsymbol{\varphi}^T\theta, \sigma^2) \tag{13}$$

For this specific case, we have

$$\nabla_\theta \log \pi(s, a) = \frac{2c}{\sigma^2}(a - \boldsymbol{\varphi}^T\theta)\boldsymbol{\varphi} \tag{14}$$

where c is a normalization constant, and the compatible advantage value function becomes

$$f_\omega^\pi(s,a) = \nabla_\theta \log \pi(s,a)^T \omega = \left[\frac{2c}{\sigma^2}(a - \boldsymbol{\varphi}^T\theta)\boldsymbol{\varphi}\right]^T \omega \tag{15}$$

Our aim here is to extend the set of basis functions $\boldsymbol{\varphi}$ by adding new basis functions, so that we would obtain better policies. In order to accomplish this goal, we need to be able to assess the effect of potential candidate basis functions on the system and choose the most promising one. One possible way to do this would be to take advantage of the inherent relationship between the policy and the compatible advantage value function. If the error in the estimation of the advantage value function can be determined, then useful basis functions such that their influence on advantage value function through the gradient of the logarithm of the policy would reduce this error, can be constructed and used to improve the policy.

Now, for a particular reinforcement learning problem, given a set of basis functions $\boldsymbol{\varphi} = \{\varphi_1(s), \ldots, \varphi_n(s)\}$ and an initial set of policy parameters θ_0, we can run episodic Natural Actor-Critic algorithm to find a sequence of natural gradient estimates and apply gradient ascent to obtain corresponding policies at each iteration. Let ω_t, θ_t, and π_{θ_t} denote the natural policy gradient, policy parameters and policy at iteration t, respectively. Note that, by definition the gradient of the logarithm of the policy for the action corresponding to the mean of the normal distribution, i.e. $\boldsymbol{\varphi}^T\theta$, in Equation 13 is the zero vector; this means that for policy π_{θ_t}, the state-action tuple $(s, \boldsymbol{\varphi}^T(s)\theta_t)$ has an advantage value of 0. Let ω be an estimation of natural policy gradient for π_{θ_t} over a set of trajectories, and $\theta = \theta_t + \epsilon\omega$ be an updated set of policy parameters, where ϵ denotes the step size. By definition, we have

$$\boldsymbol{\varphi}^T(s)\theta = \boldsymbol{\varphi}^T(s)\theta_t + \epsilon\boldsymbol{\varphi}^T\omega$$

that is the mean of the normal distribution moves by an amount of $\epsilon\boldsymbol{\varphi}^T\omega$ at state s. By putting $\boldsymbol{\varphi}^T(s)\theta$ in Equation 15, we obtain an advantage difference of

$$f_\omega^\pi(s, \boldsymbol{\varphi}^T(s)\theta) = \epsilon\boldsymbol{\varphi}^T(s)\omega\boldsymbol{\varphi}^T(s)\omega_t$$

which can be used as an estimate error in the advantage value of the state-action tuple $(s, \boldsymbol{\varphi}^T(s)\theta_t)$. Once we have the error estimates, we can then employ cascade correlation learning in order to construct the basis functions. Let $\mathcal{N}$ be a cascade correlation network with n inputs, where n is the number of initial basis functions, and a single output. We will not be using the output node, but only input and hidden nodes, as the parameters of the policy and the compatible advantage value function are determined by the eNAC algorithm. Initially, the network does not have any hidden neurons and all input neurons are directly connected to the output neuron; the activation function of the i^{th} input neuron is set φ_i, i.e. when fed with state s they output $\varphi_i(s)$. For each sample of the given set of trajectories, in the cascade correlation network we input s to the input nodes, set $\epsilon\boldsymbol{\varphi}^T(s)\omega\boldsymbol{\varphi}^T(s)\omega_t$ as the residual output error, and train candidate units that are highly correlated with the residual output error. Note that, in Equation 15 $(a-\boldsymbol{\varphi}^T\theta)$ term is common for all basis functions, and hence there is a linear dependency between the candidate units and the compatible advantage value function. At the end of the training phase, the candidate unit having the maximum correlation is added to the network by transforming it into a hidden neuron and becomes the new basis function φ_{n+1}; $\varphi_{m+1}(s,a)$ can be calculated by feeding s as input to the network and determining the activation value of the hidden neuron. The parameter vectors θ and ω are also expanded. By continuing eNAC training, one can calculate the natural policy gradient and update the current policy, which now is represented by one more basis functions and potentially more powerful. This process can be repeated at certain intervals, introducing a new basis function at time, until a policy with adequate performance is obtained or new functions do not improve the exiting policy. In this hybrid learning system, the basis functions are determined by the cascade correlation network, and the corresponding parameters of the policy and the advantage value function are regulated by the natural actor-critic algorithm. Note that, the values of all basis functions for a given (s,a) tuple can be found with a feed-forward run over the network, and as stated before can be cached for efficiency reasons if desired.

4 Experiments

We have evaluated the proposed method on a new variant of the well known cart-pole problem, called spring cart-pole. Spring cart-pole is a dynamica system where the state is defined by the position and velocity of the elements of the system, and actions (applied forces) define the next state: it is a non-linear control task with continuous state spaces. In spring cart-pole, there are two carts, instead of one in the original version, that are connected to each other by a spring. The spring restricts the movement of carts with respect to each other, and makes the problem more complex. Both carts try to balance their own poles in upright position while staying on the track and staying close to each other. An episode ends if one of the carts move out of the boundary of track, move very close to or far away from the other cart. The eight state variables are the angles of the poles and their angular speed, and the position of the carts and their velocity. The reward is equal to sum of the cosine of the angle of the poles. Forces that can be applied to the carts are limited to [-2,2].

In cascade correlation network, we trained an equal number of candidate units having Gaussian and sigmoid activation functions using RPROP method [13]. In RPROP, instead of directly relying on the magnitude of the gradient for the updates (which may lead to slow convergence or oscillations depending on the learning rate), each parameter is updated in the direction of the corresponding partial derivative with an individual time-varying value. The update values are determined using an adaptive process that depends on the change in the sign of the partial derivatives. We allowed at most 100 passes over the sample set during

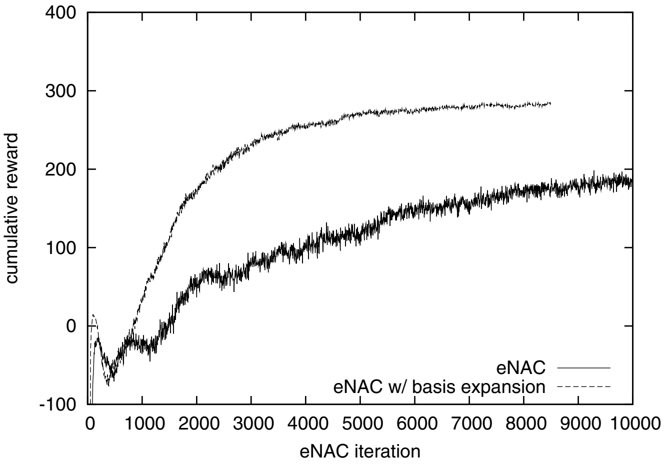


Fig. 2. eNAC using original state variables vs. eNAC with basis function expansion. A new basis function is added after every 500 iterations, up to a number of 16. After 10 basis functions, the policy converges to a near optimal policy.

the training of candidate units, and employed the following parameters: $\triangle_{min} = 0.0001, \triangle_{ini} = 0.01, \triangle_{max} = 0.5, \eta^{-} = 0.5, \eta^{+} = 1.2$.

Figure 2 shows the training results for the spring cart-pole problem averaged over 40 runs. The policy parameters are updated by calculating the natural policy gradient using 40 trajectories. For the simulations, we chose a rather complex initial state in which the poles are in downright position, the first cart is stationed in the middle of the track, and the second cart is half-way from it. Each trajectory starts from a slightly perturbed initial state, and is limited to a length of 200 steps. The initial policy parameters are set to 0. By employing the proposed method, a new basis function is trained every 500 policy update using the cascade correlation learning algorithm, and added to the set of basis functions of the policy. Every 5 policy update, we made a test run of 500 time steps to measure the performance of the current policy. During testing the variance of the policy distribution is set to 0, i.e. stochasticity is removed. As can be seen from the figure, policies represented by the automatically constructed basis functions outperform those that use the original basis functions that fell short of attaining near-optimal levels. Note that, the performance of the policies increase steadily, starting from as early as two additional basis functions, which indicates that the proposed method is successful in constructing useful basis functions from the existing ones.

5 Related Work

Basis function, or feature selection and generation is essentially an information transformation problem; the input data is converted into another form that "better" describes the underlying concept and relationships, and "easier" to process by the agent. As such, it can be applied as a preprocessing step to a wide range of problems and have been in the interest of the data-mining community, in particular for classification tasks (see [14]). Following the positive results obtained using efficient methods that rely on basis functions (mostly, using linear approximation architectures) in various domains, it also recently attracted attention from the RL community. However, the existing research is focused on constructing basis functions for approximating the value function, and, contrary to our work presented in this paper, do not consider the direct policy representation.

In [15], Menache et al. examined adapting the parameters of a fixed set of basis functions (i.e, center and width of Gaussian radial basis functions) for estimating the value function of a fixed policy. In particular, for a given set of basis function parameters, they used $LSTD(\lambda)$ to determine the weights of basis functions that approximate the value function of a fixed control policy, and then applied either a local gradient based approach or global cross-entropy method to tune the parameters of basis functions in order to minimize the Bellman approximation error in a batch manner. The results of experiments on a grid world problem show that cross-entropy based method performs better compared to the gradient based approach.

In [16], Keller et al. studied automatic basis function construction for value function approximation within the context of LSTD. Given a set of trajectories and starting from an initial approximation, they iteratively use neighborhood component analysis to find a mapping from the state space to a low-dimensional space based on the estimation of the Bellman error, and then by discretizing this space aggregate states and use the resulting aggregation matrix to derive additional basis functions. This tends to aggregate states that are close to each other with respect to the Bellman error, leading to a better approximation by incorporating the corresponding basis functions.

In [17], Parr et al. showed that for linear fixed point methods, iteratively adding basis functions such that each new basis function is the Bellman error of the value function represented by the current set of basis functions forms an orthonormal basis with guaranteed improvement in the quality of the approximation. However, this requires that all computations are exact, in other words, are made with respect to the precise representation of the underlying MDP. They also provide conditions for the approximate case, where progress can be ensured for basis functions that are sufficiently close to the exact ones. A new basis function for each action is added at each policy-evaluation phase by directly using locally weighted regression to approximate the Bellman error of the current solution.

In contrast to these approaches that make use of the approximation of the Bellman error, including ours, the work by Mahadevan *et al.* aims to find policy and reward function independent basis functions that captures the intrinsic domain structure that can be used to represent any value function [18,19,20]. Their approach originates from the idea of using manifolds to model the topology of the state space; a state space connectivity graph is built using the samples of state transitions, and then eigenvectors of the (directed) graph Laplacian with the smallest eigenvalues are used as basis functions. These eigenvectors possess the property of being the smoothest functions defined over the graph and also capture the nonlinearities in the domain, which makes them suitable for representing smooth value functions.

To the best of our knowledge, the use of cascade correlation networks in reinforcement learning has rarely been investigated before. One existing work that we would like to mention is by Rivest and Precup (2003), in which a cascade correlation network together with a lookup-table is used to approximate the value function in an on-line temporal difference learning setting [21]. It differs from our way of utilizing the cascade correlation learning architecture to build basis functions in the sense that in their case, cascade correlation network purely functions as a cache and an approximator of the value function, trained periodically at a slower scale using the state-value tuples stored in the lookup-table.

6 Conclusion

In this paper, we explored a new method that combines cascade correlation learning architecture with episodic Natural-Actor Critic algorithm to find a set of basis function that would lead to a better and more expressive representation for the policy, and consequently results in improved performance. The experimental results indicate that it is effective in discovering such functions. An important property of the proposed method is that the basis function generation process requires little intervention and tuning from the user.

We think that learning sparse representation for states in a very important issue to tackle large reinforcement learning problems. A lot of work is still due to get a proper, principled approach to achieve this, not mentioning the theoretical issues that are pending. We pursue future work in this direction and also apply the method to more complex domains.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998); A Bradford Book
2. Howard, R.: Dynamic programming and Markov processes. MIT Press, Cambridge (1960)
3. Puterman, M.: Markov Decision Processes — Discrete Stochastic Dynamic Programming. Probability and mathematical statistics. Wiley, Chichester (1994)
4. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Neural Information Processing Systems (NIPS), pp. 1057–1063 (1999)
5. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256 (1992)
6. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM J. Control Optim. 42(4), 1143–1166 (2003)

7. Peters, J., Schaal, S.: Policy gradient methods for robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2006, pp. 2219–2225 (2006)
8. Amari, S.i.: Natural gradient works efficiently in learning. Neural Computation 10(2), 251–276 (1998)
9. Peters, J., Schaal, S.: Natural actor-critic. Neurocomput. 71(7-9), 1180–1190 (2008)
10. Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M.: Incremental natural actor-critic algorithms. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S. (eds.) Advances in Neural Information Processing Systems, vol. 20, pp. 105–112. MIT Press, Cambridge (2008)
11. Riedmiller, M., Peters, J., Schaal, S.: Evaluation of policy gradient methods and variants on the cart-pole benchmark. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007, pp. 254–261 (2007)
12. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems. Denver 1989, vol. 2, pp. 524–532. Morgan Kaufmann, San Mateo (1990)
13. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the rprop algorithm, vol. 1, pp. 586–591 (1993)
14. Guyon, I., Elisseff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (2003)
15. Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research 134, 215–238 (2005)
16. Keller, P.W., Mannor, S., Precup, D.: Automatic basis function construction for approximate dynamic programming and reinforcement learning. In: ICML, pp. 449–456. ACM, New York (2006)
17. Parr, R., Painter-Wakefield, C., Li, L., Littman, M.: Analyzing feature generation for value-function approximation. In: ICML, pp. 737–744. ACM, New York (2007)
18. Mahadevan, S.: Representation policy iteration. In: UAI, pp. 372–379 (2005)
19. Johns, J., Mahadevan, S.: Constructing basis functions from directed graphs for value function approximation. In: ICML, pp. 385–392. ACM, New York (2007)
20. Mahadevan, S., Maggioni, M.: Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Journal of Machine Learning Research 8, 2169–2231 (2007)
21. Rivest, F., Precup, D.: Combining TD-learning with cascade-correlation networks. In: Fawcett, T., Mishra, N. (eds.) ICML, pp. 632–639. AAAI Press, Menlo Park (2003)

Reinforcement Learning with the Use of Costly Features

Robby Goetschalckx[1], Scott Sanner[2], and Kurt Driessens[1]

[1] Declarative Languages and Artificial Intelligence,
Katholieke Universiteit Leuven, Leuven, Belgium
{robby,kurtd}@cs.kuleuven.be
[2] National ICT Australia
Scott.Sanner@nicta.com.au

Abstract. In many practical reinforcement learning problems, the state space is too large to permit an exact representation of the value function, much less the time required to compute it. In such cases, a common solution approach is to compute an approximation of the value function in terms of state features. However, relatively little attention has been paid to the cost of computing these state features. For example, search-based features may be useful for value prediction, but their computational cost must be traded off with their impact on value accuracy. To this end, we introduce a new cost-sensitive sparse linear regression paradigm for value function approximation in reinforcement learning where the learner is able to select only those costly features that are sufficiently informative to justify their computation. We illustrate the learning behavior of our approach using a simple experimental domain that allows us to explore the effects of a range of costs on the cost-performance trade-off.

1 Introduction

We examine cost-sensitive linear-value function approximation in a reinforcement learning context where certain state features are only available at a certain cost. This cost could reflect time or other resources spent on the process of acquiring the feature information, but we assume that this cost can be transformed into the same units used to represent reward in the original reinforcement learning problem.

As a motivating example, consider an agent playing a game of perfect information such as Backgammon or Othello where the opponent executes a stationary policy. The agent knows the rules and thus has access to an accurate model of the environment, except for the opponent policy, which we assume to be unknown. While any reinforcement learning problem of this nature can be solved in theory by using an exact enumerated-state representation of the value function, this is often infeasible in practice due to time and space constraints. Thus, we must often resort to techniques for computing an approximation of the value function in terms of state features.

While value function approximation is well-addressed in the reinforcement learning literature (c.f. Chapter 8 of [1]), the cost of feature computation is often considered negligible and thus ignored. However, continuing our game-playing example, we note that costly search-based state features may be useful for predicting the value of a state. For instance, a useful state feature in a game might be the result of an n-ply expected minimax search. However, there will often be a limit on the time available for a game player to make decisions – either for the entire game or per turn – after which the game is forfeited. In such a setup, it is important to find a good trade-off between the cost of computation necessary to make a decision and the quality of the resulting decision.

Various theoretical approaches are possible to model this trade-off. While the original optimal reinforcement learning problem we consider can be modelled as a Markov decision process (MDP), one might consider modelling the function approximation setting as a partially observable MDP (POMDP) [2], using information-gathering actions to represent the computation of costly features. In theory, an optimal policy for this POMDP would select those features to compute at any decision stage in order to optimally trade-off feature cost w.r.t. its impact on future reward. However, such a framework requires embedding an already difficult-to-solve MDP inside a POMDP and then solving that POMDP.[1] Such an approach will not generally be feasible in practice.

Here we propose a more pragmatic approach where we learn the relative value of features in an explicit way. To do this, we approximate the value function using cost-sensitive sparse linear regression techniques, trading off prediction errors with the costs induced by using a feature. While such an approach does not guarantee that the optimal set of features will be chosen at any decision-stage w.r.t. the cost-performance trade-off, it does guarantee that the prediction accuracy will improve by at least the total cost of the features used.

2 Related Work

Cost-sensitive regression and cost-sensitive classification in non-sequential decision making, supervised learning settings have been widely studied, for example in [3]. While [4] addresses one aspect of cost-sensitive sequential decision-making, it should be noted that in their special case, cost is only associated with actions – not with observing state features – so standard reinforcement learning techniques can be applied without modification. In our work, we specifically consider the case of reinforcement learning with function approximation where state features are costly to compute. This induces a more difficult problem in that standard reinforcement learning techniques must be modified to trade off the cost of using a feature w.r.t. its impact on prediction accuracy during value approximation.

Other work which handles reinforcement learning with costs is discussed in [5]. Here the costly features are considered to have binary values, which allows for the construction of a cost-sensitive regression tree (related to a cost-sensitive classification using a decision tree) where the value of a computed feature can be used to decide which other features to use. In contrast, here we consider the case of linear-value function approximation with real-valued costly features. In the presence of both binary and real-valued features, these two approaches could be merged although such extensions are beyond the scope of the current paper.

The value of information has been formalized by Howard [6] and can be used as a framework to estimate the expected utility increase of observing a random variable (e.g., a feature) given prior information. The meta-reasoning paradigm [7] extends this idea

[1] This is only one difficulty with the POMDP approach. See Section 2 for further discussion.

to sequential decision-making by trading off the allocation of computational resources over time w.r.t. the expected gain of using those resources. The difference between the meta-reasoning paradigm and our work is that we do not (and for all practical purposes, cannot) directly model the predictions of costly features since the ability to accurately model them would preclude the need to actually compute them.

When an MDP has costly-observable state (i.e., not state features, but the underlying state itself), the problem may be formally modeled as a POMDP. Variants of such approaches are explored in [8] and [9]. However, such problems are inherently more difficult than the case we consider here. In our case, the underlying problem we are trying to solve is an MDP with fully observable state (zero-cost, by definition), not a POMDP. Our difficulties with costly features only arise when considering the value function approximation paradigm. While we could use a POMDP model to formalize the decision-theoretic trade-off between feature computation and prediction accuracy in this approximation, such an approach would be impractical: Not only would the POMDP be intractably large to solve, without a model of information-gathering actions[2], the POMDP solution would only become further complicated by the need to perform belief-state updating on a model of such actions from experience. While analytical solutions to related problems exist in theory (c.f., [10]), such approaches are incapable of practically scaling beyond all but the smallest problems.

3 Reinforcement Learning with Costly Features

In this section, we review the general framework of function approximation in reinforcement learning and then proceed to describe our modifications to accommodate costly features.

3.1 MDPs and Reinforcement Learning

We assume the decision-making environment to be a *Markov decision process* (MDP) [11] with which an agent interacts by repeatedly executing an action in the current state, receiving a reward signal and then stochastically transitioning to a successor state. Formally, an MDP can be defined as a tuple $\langle S, A, T, R, \gamma \rangle$. $S = \{s_1, \ldots, s_n\}$ is a finite set of fully observable states. $A = \{a_1, \ldots, a_m\}$ is a finite set of actions. $T : S \times A \times S \rightarrow [0, 1]$ is a stationary, Markovian transition function. We often express T as the conditional probability distribution $P(s'|s, a)$. We will assume that a reward $R : S \times A \rightarrow \mathbb{R}$ is associated with every state and action. γ is a discount factor s.t. $0 \leq \gamma < 1$[3] used to specify that a reward obtained t timesteps into the future is discounted by γ^t.

A policy $\pi : S \rightarrow A$ specifies the action $a = \pi(s)$ to take in each state s. The value $Q^\pi(s, a)$ of taking an action a in state s and then following the policy π thereafter can be defined using the infinite horizon, expected discounted reward criterion:

[2] If an accurate model of a costly information-gathering action existed, it could be substituted in place of the action itself to obtain an equivalent zero-cost action.

[3] With modifications to enforce that total accumulated reward is finite, $\gamma = 1$ could be accommodated.

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\cdot r_{t}\middle| s_0 = s, a_0 = a\right] \tag{1}$$

where r_t is the reward obtained at time t (assuming s_0 and a_0 respectively represent the state and action at $t = 0$). Then we can define a value function $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$ that represents the expected value obtained by starting in state s and acting according to π.

The objective in an MDP is to find a policy π^* such that $\forall \pi, s.\ V^{\pi^*}(s) \geq V^{\pi}(s)$. At least one such optimal policy is guaranteed to exist and, in addition, the following Bellman optimality equation is known to hold for π^* [11]:

$$V^{\pi^*}(s) = max_{a\in A}\left\{R(s,a) + \gamma \cdot E_{\pi^*}\left[V^{\pi^*}(s_{t+1})\middle| s_t = s\right]\right\}$$

In the *reinforcement learning* setting, the transition and reward model may not be explicitly known to the agent although they both can be sampled from experience. Here, we assume the *generalized policy iteration* (GPI) framework that is known to capture most reinforcement learning approaches [1]. GPI interleaves policy evaluation and policy update stages as follows:

```
Generalized Policy Iteration (GPI)

1. Start with arbitrary initial policy $\pi_0$ and set $i = 0$.
2. Estimate $Q^{\pi_i}(s,a)$ from experience (e.g., using Equation 1).
3. Let $\pi_{i+1}(s) = \arg\max_{a\in A} Q^{\pi_i}(s,a)$.
4. If termination criteria not met, let $i = i + 1$ and goto step 2.
```

Every reinforcement learning algorithm that is an instance of the above GPI algorithm may prescribe its own method for performing each step and many such instances are known to have strong convergence guarantees. For now, we keep our treatment of reinforcement learning with costly features as general as possible. Specifically, this means that in the context of GPI, we can restrict our discussion of reinforcement learning with costly features to Q-value estimation.

3.2 Cost-Sensitive Value Approximation

In practice, it is often infeasible to work with an enumerated state representation due to time and space constraints. A common solution approach in this case is to resort to value function approximation in step 2 of the GPI algorithm by defining relevant state features. In this case, the agent does not directly observe the exact state s of the environment. Instead, the agent has access to a set of state features $F = \{f_1, \ldots, f_k\}$, where for each $f \in F$, $f : S \to \mathbb{R}$ is an (apriori unknown) mapping from a state to $\mathbb{R}$. The benefits of this approach are well-known: (1) an accurate approximation of a Q-function (and thus implicitly, a policy) can often be represented with $|F| << |S|$ and (2) a limited set of descriptive features enables *generalization* of learned value across multiple states, leading to faster learning.

However, as argued in Section 1 for the games setting, it is plausible to consider using costly features such as those that perform search. Thus, we assume that each feature f is associated with a cost function $c_f : S \rightarrow \mathbb{R}$, which represents the cost of computing feature f in state s.[4] We assume that the feature cost functions c_f and the reward R are expressed in the same units. For a game where there is a fixed time available per move (after which the game is forfeited), a feature cost could be set to a fraction of the loss value corresponding to the time spent computing the feature.

In the setting of value function approximation with costly features, we must modify our MDP solution criterion to consider both the original reward signal as well as the cost of computing a particular choice of features F. To facilitate this modification, we introduce a new meta-policy $\Pi = \langle \pi, \mathcal{F} \rangle$ where π is the policy for the original MDP and $\mathcal{F} : F \times S \times A \rightarrow \{true, false\}$ is a feature selection function that indicates which features should be selected when evaluating the Q-value for a given state and action. Abusing notation slightly, we often use $\mathcal{F}(s,a)$ to directly denote the subset of features $F' \subseteq F$ selected for a given state s and action a.

However, there is one additional and important complication to value function approximation with costly features. Since we do not represent the policy explicitly, but rather implicitly by evaluating a set of Q-functions, we must take into account the cost of evaluating a set of Q-functions for policy π w.r.t. our feature selection criterion $\mathcal{F}$. In light of this issue and our previous definitions, we now formally define our problem:

Definition 1 (Cost-sensitive Value Approximation). Given *an MDP* $\langle S, A, T, R, \gamma \rangle$, *a set of state features* F *and their related cost functions* c_f *and a policy* π.

Find *a feature selection function* $\mathcal{F}^*$ *for meta-policy* $\Pi^* = \langle \pi, \mathcal{F}^* \rangle$, *such that*

$$\mathcal{F}^* = \arg\max_{\mathcal{F}} \left\{ E_{s \sim P(s|\pi,\mathcal{F})} \left[V^{\langle \pi, \mathcal{F} \rangle}(s) \right] \right\} \tag{2}$$

where $P(s|\pi,\mathcal{F})$ *is a state occupancy distribution induced for meta-policy* $\langle \pi, \mathcal{F} \rangle$ *and*

$$V^{\langle \pi, \mathcal{F} \rangle}(s) = E_{\langle \pi, \mathcal{F} \rangle} \left[\sum_{t=0}^{\infty} \gamma^t \left(r_t - \sum_{a'_t \in A} \sum_{f \in \mathcal{F}(s_t, a'_t)} c_f(s_t, a'_t) \right) \middle| s_0 = s \right] \tag{3}$$

In words, $V^{\langle \pi, \mathcal{F} \rangle}(s)$ represents the infinite-horizon discounted reward starting from state s and following policy π thereafter. In addition to the reward r_t accumulated at time t, this value definition also explicitly models the cost of computing the meta-policy at time t via the cost of computing a Q-function for each action a'_t w.r.t. $\mathcal{F}(s_t, a'_t)$. The objective itself is to find the feature selection function $\mathcal{F}$ that maximizes the value $V^{\langle \pi, \mathcal{F} \rangle}(s)$ for each state s, weighted by the occupancy probability of s w.r.t. the meta-policy.

[4] We can easily relax both the feature mapping and its cost function to be stochastic (e.g., for randomized search-based features) since our reinforcement learning approach is sample-based. We provide the deterministic case here to simplify our notation and presentation.

Perhaps one of the most interesting observations about value function approximation with costly features is that even though π is assumed to be fixed, the actual policy executed varies according to $\mathcal{F}$. This occurs because in the function approximation setting, the policy is computed implicitly w.r.t. Q-values, which are themselves modulated by a feature selection function $\mathcal{F}$. In this sense, it appears quite difficult to optimally compute Definition 1 without exhaustive enumeration of all possible $\mathcal{F}$ thus requiring $2^{|F|}$ evaluations. This is intractable for sufficiently large $|F|$ and thus we focus on approximate solutions with weaker optimality guarantees in the next section.

4 Sparse Linear-Value Approximation

So far we have not assumed a specific functional form for our value approximation. However, from here out, we focus solely on linear value-approximation techniques, not only because linear regression is one of the most widely used and well-understood function approximation methods, but also because it admits elegant sparse solutions that will be useful in minimizing feature usage, and thus feature cost.

We represent a value approximation $\hat{V}_{\boldsymbol{w}}^{\langle\pi,\mathcal{F}\rangle}(s)$ of $V^{\langle\pi,\mathcal{F}\rangle}(s)$ from Equation 3 as a linear combination of features with weights $\boldsymbol{w} = \langle w_0, \ldots, w_k \rangle$ with each $w_i \in \mathbb{R}$:

$$\hat{V}_{\boldsymbol{w}}^{\langle\pi,\mathcal{F}\rangle}(s) = w_0 + \sum_{f_i \in \mathcal{F}(s)} w_i f_i(s) \tag{4}$$

Here $\mathcal{F}(s)$ is not considered to be action-dependent and thus we drop the action argument.

When a policy cannot be derived directly from a value function, we could use action-dependent weights $\boldsymbol{w}_a$ for all $a \in A$ to learn a Q-value approximation for each action:

$$\hat{Q}_{\boldsymbol{w}}^{\langle\pi,\mathcal{F}\rangle}(s,a) = w_{a,0} + \sum_{f_i \in \mathcal{F}(s,a)} w_{a,i} f_i(s) \tag{5}$$

For simplicity, we focus on direct value approximation in the following presentation although the framework can be easily modified to handle Q-value approximation as well.

4.1 Least Angle Regression Methods

Due to their sparsity properties, we focus on a class of linear regression techniques collectively referred to as *least-angle regression* (LAR) methods, such as *lasso* and *forward stagewise regression* [12].

In the context of linear-value approximation, LAR methods provide a solution to the following linear regression problem with cost budget C where we define the indicator function $\mathbb{I}[\cdot]$ to take the value 1 when its argument $\cdot$ is true and 0 otherwise:

$$\begin{aligned} \text{Minimize: } & \sum_s P(s|\pi,\mathcal{F}) \left[\hat{V}_{\boldsymbol{w}}^{\langle\pi,\mathcal{F}\rangle}(s) - V^{\langle\pi,\mathcal{F}\rangle}(s) \right]^2 \\ \text{Subject to: } & \left[\sum_s P(s|\pi,\mathcal{F}) \sum_{i=0}^{k} (\alpha|w_i| + \mathbb{I}[w_i \neq 0] \cdot c_{f_i}(s)) \right] \leq C \end{aligned} \tag{6}$$

Although notationally cumbersome, this optimization problem simply states that we wish to minimize the sum-of-squared errors of the approximate value function $\hat{V}_{\boldsymbol{w}}^{\langle \pi, \mathcal{F} \rangle}(s)$ w.r.t. samples from the true distribution $V^{\langle \pi, \mathcal{F} \rangle}(s)$ weighted by state occupancy probability. The constraints simply state that the total sum of weights and feature costs for non-zero weights should be less than C. Here, α is a constant indicating how important weight regularization is relative to minimizing the costs. For ordinary LAR, this value is equal to 1 (while the costs are zero). For our preliminary experiments as presented in section 5, we chose $\alpha = 0$.

At first, the budget C would seem to be an unnecessary since the feature costs are already accounted for in the value function estimate. However, including this additional constraint has two advantages: (1) the use of small budgets C encourage sparsity in the weights, thereby maximizing the predictive power of the subset of features with non-zero weights, and (2) by incrementally increasing C from 0 to ∞, we can greedily add in new features from F, thus providing us with an efficient way to explore the entire feature selection function without enumerating all possibilities.

Fortunately, the forward-stagewise regression solution to optimizing the above problems provides us with an efficient way of doing exactly this. We briefly describe this below and refer the reader to the detailed discussion in [12] for more information on this and related methods. Our adaptation of the forward-stagewise regression algorithm, dubbed FOVEA, can be seen in Fig. 1.

Forward-stagewise Value Approximation (FOVEA)

1. Normalize all feature predictions in F to have 0 mean and a variance of 1.
2. Initialize the step-size η to some small positive value.
3. Initialize $\mathcal{F}$ such that $\forall s. \mathcal{F}(s) = \emptyset$.
4. Given a policy π and current $\mathcal{F}$, collect a batch of data samples $V^{\langle \pi, \mathcal{F} \rangle}(s)$ by executing meta-policy $\langle \pi, \mathcal{F} \rangle$ (this implicitly generates samples with state distribution $P(s|\pi, \mathcal{F})$), initialize w_0 with the average value of the batch (this gives the residuals a mean of 0), and repeat the following:
   (a) calculate the correlation score (absolute correlation with the current residual $r(s)$, *minus the cost* if the feature is not yet included in the linear function) for every feature:
   $$score_i = \left| \sum_s f_i(s) r(s) \right| - (1 - \delta_i) \sum_s cost_i(s)$$
   (here δ_i indicates whether the feature is already included in the sum: if $f_i \in \mathcal{F}$, $\delta_i = 1$, otherwise $\delta_i = 0$).
   (b) Find the feature f_i with the highest score, and with $score_i \geq 0$, if no such feature found, halt algorithm
   (c) If the f_i was not yet included in $\mathcal{F}$, let $F = F \cup \{f_i\}$
   (d) Increment or decrement w_i by η to reduce the residuals.

Fig. 1. The FOVEA Algorithm

Note that we do not normalize the target values, which is the normal procedure for least-angle regression. Our reason for this is that by not dividing by the standard deviation, the residuals at each step correspond to the actual errors. This is necessary for the trade-off of cost and error.

It should be noted that the forward stage-wise approach is a *greedy* selection approach. Because of this, the approximation obtained might not be the optimal one in all cases. A local optimum is guaranteed, however and the increase in value prediction accuracy of using this greedy feature set is guaranteed to equal or exceed the cost of its computation.

5 Experiments

A first setting we used for experiments is a simple deterministic corridor domain (figure 2). The state space consists of five rooms, labeled $s_1, \ldots, s_5$. From each state two actions, $+1, -1$ can be performed. Performing action $+1$ in state s_i for $i < 5$ leads to s_{i+1} and performing -1 in state s_i for $i > 1$ leads to s_{i-1}. All other actions take the agent to the center s_3. A reward of 1 is assigned for taking $+1$ in s_5 and -1 is assigned for taking action -1 in s_1. All other rewards are equal to 0. We used a discount factor $\gamma = 0.9$.

We provided seven state-action indicator features f_i for $1 \leq i \leq 7$ to the agent where taking action $a \in \{+1, -1\}$ in i results in $f_{i+a} = 1$ with all remaining indicator features set to 0. f_0, f_2, f_3, f_5 and f_6 are free while f_1 and f_4 have a cost c. Furthermore two random number generators were provided to the agent, one which was free and another one which had a high cost 0.5. Finally, the state-action feature indicators $f_0, \ldots, f_6$ were copied but now with the higher cost 0.5. We used FOVEA to approximate Q-values using the state-action features defined above. We used 100 samples for each update unless stated otherwise. All results shown are averages over 10 runs.

A first experiment was performed with $c = 0$. This was an initial check to verify that out algorithm works as expected. Indeed, the agent always learned to use the free, informative features, and never to use the random features or the costly ones.

In a second experiment we varied the value of c over a range of 0 to 0.5. Increasing c takes away the possibility for the agent to locate itself exactly without paying any cost: if the agent does not pay c, it can not distinguish between x_1 and x_4. (Paying for only f_1 or f_4 is enough, however: if the agent knows he's not in one of the freely observable

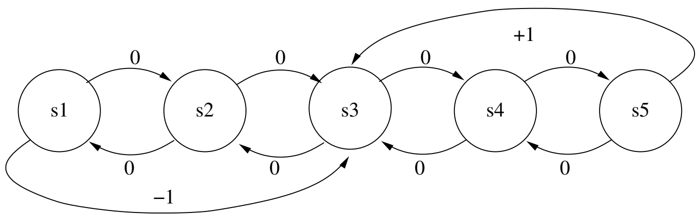


Fig. 2. A simple domain with five linked states. Actions and their corresponding transitions are labeled with their reward.

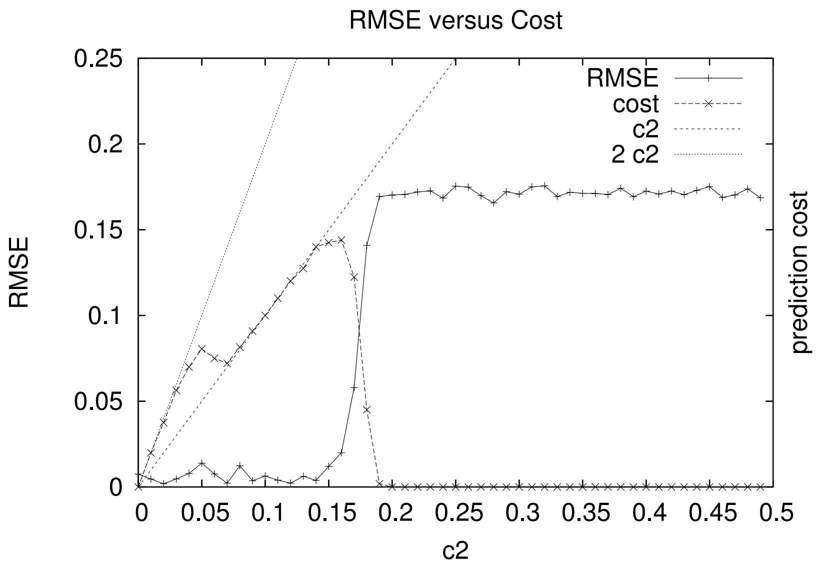


Fig. 3. Error the agent is aware of making versus the amount spent on costly features

states, and not in x_4, he must be in x_1 and vice versa.) We predicted that there is a certain threshold-value for the cost where the agent will be undecided on using one of the features f_1 or f_4 and paying the cost or not using these. If c is much lower, the agent will always use one of the features, if it is much higher it will not pay off to spend the cost in exchange for the information. For the forward-selection approach, this is exactly what happened with a clear-cut phase transition near $c = 0.185$. This is actually the theoretically correct value for the threshold in this problem domain. This can be clearly seen in figure 3.

Here the cost the agent pays is compared with the prediction error the agent is aware of making (the root mean squared error, or the 2-norm of the final residual). For very low values of c, the agent actually computes both of the features while only one is needed. As the costs are indeed very low, this does not pose a real problem. Using larger batches to compute the new linear regression function would remedy this problem. For low values of c, the agent keeps paying for one of the features (so the spent cost is equal to c). When getting close to the threshold value of 0.185, however, the agent is no longer willing to pay for the information. Indeed, for values higher than the threshold, the error the agent is aware of making is lower than the cost of extra information. Given longer training time and larger batches for the updates, the phase transition would become even more clear.

For varying values of c the root mean square error of the predictions (compared to the actual optimal values) as the number of examples increases is shown in figure 4. For the value of 0.185 only in about half of the runs the agent was deeming it worth the investment. As this is the threshold value, for which the agent should in theory be undecided whether or not to pay for the extra information, this is what was expected. For lower

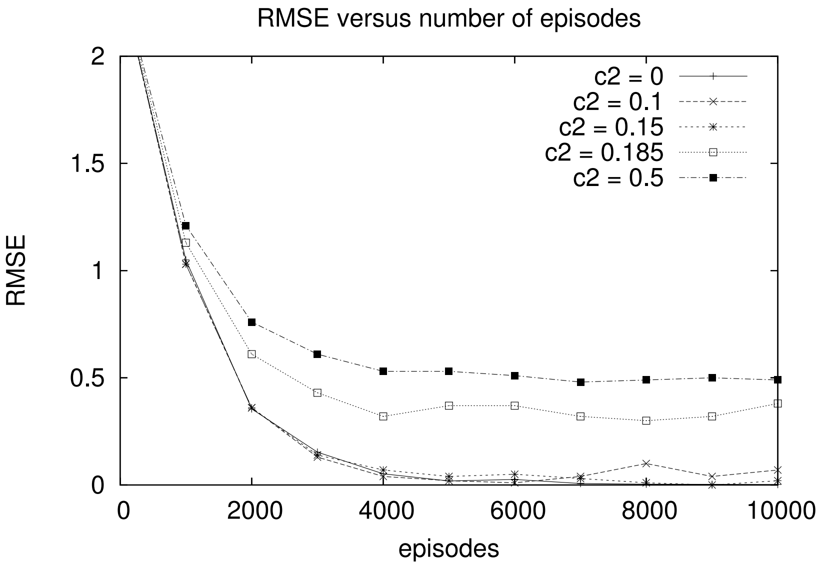


Fig. 4. Evolution of the average prediction error compared to the ideal solution as the number of episodes increases

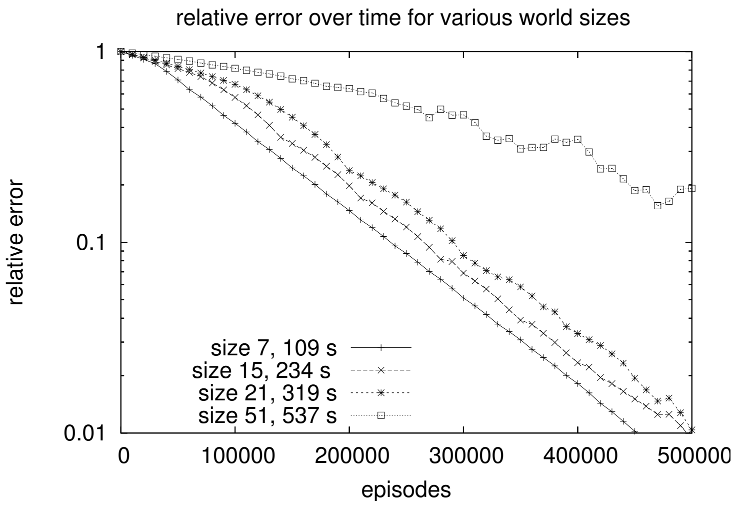


Fig. 5. Relative error on the predictions for varying world sizes

values than this, the agent quickly learns that the feature is worth its cost. (In figure 4 results are shown for every 1000th episode. During the first 1000 episodes, there is a slight difference for the lower costs, with faster convergence when the information is cheaper.)

For a third experiment the size of the domain was increased to see how the algorithm scales up. The domain still consists of a corridor, with a positive and negative reward at respectively the rightmost and the leftmost room. The features provided were state-action indicators as before. Each indicator had a cost inversely proportional to the world size (this compensates for the inherently lower differences in value function for neighboring states). In figure 5 one can see the sum of the relative errors on the predictions of all state values as the number of episodes increases (note the logarithmic scale on the vertical axis). The value approximation was updated each 10000 samples.

From this figure it is clear that for all these world sizes there was similar convergence behavior. For the larger world (size 51), the convergence is slower, which is understandable, as it takes more time to propagate updates over the entire domain. In the legend of figure 5 the run-times for these domains is also shown. As predicted, the runtime is about linear in the number of features, in this case the size of the world.

6 Conclusions and Future Work

Faced with the task of value approximation in reinforcement learning with costly features, we introduced a novel sparse linear-value approximation approach to efficiently select the features for value prediction that are sufficiently informative to justify their computation. In experimentation, our forward-stagewise value approximation algorithm provided near-perfect trade-offs in value prediction exhibiting sharp phase transitions at theoretical switchover points where feature computation no longer paid-off in reward gain. Furthermore, our experiments demonstrated the ability of our approach to scale in terms of performance and training samples over a range of problem sizes.

While we could only provide an initial investigation of reinforcement learning with costly features in this work, our results warrant future experimentation on larger more difficult problems such as game domains with search-based features. In addition to such experimental evaluation, various efficiency enhancements can be explored in the forward-stagewise regression framework to avoid discarding samples every time the feature selection function is updated. Altogether, such advances should make possible a new and useful paradigm for large-scale reinforcement learning in real-world domains where useful features cannot always be assumed to be cost-free.

Acknowledgements

This research was sponsored by the fund for scientific research (FWO) of Flanders, of which Kurt Driessens is a postdoctoral fellow, and by National ICT Australia.

References

1. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (1998)
2. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101, 99–134 (1998)
3. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, pp. 155–164 (1999)

4. Pednault, E., Abe, N., Zadrozny, B.: Sequential cost-sensitive decision making with reinforcement learning. In: KDD 2002: Proceedings of the International Conference on Knowledge discovery and data mining, pp. 259–268. ACM, New York (2002)
5. Goetschalckx, R., Driessens, K.: Cost sensitive reinforcement learning. In: Kuter, U., Aberdeen, D., Buffet, O., Stone, P. (eds.) Proceedings of the workshop on AI Planning and Learning, pp. 1–5 (2007)
6. Howard, R.A.: Information value theory. IEEE Transactions on Systems Science and Cybernetics SSC-2, 22–26 (1966)
7. Russell, S., Wefald, E.: Principles of metareasoning. Artificial Intelligence 49 (1991)
8. Zubek, V.B., Dietterich, T.G.: A POMDP approximation algorithm that anticipates the need to observe. In: Pacific Rim International Conference on Artificial Intelligence, pp. 521–532 (2000)
9. Fox, R., Tennenholtz, M.: A reinforcement learning algorithm with polynomial interaction complexity for only-costly-observable mdps. In: AAAI, pp. 553–558 (2007)
10. Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete bayesian reinforcement learning. In: ICML 2006: Proceedings of the 23rd international conference on Machine learning, pp. 697–704. ACM, New York (2006)
11. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
12. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Technical report, Statistics Department, Stanford University (2002)

Variable Metric Reinforcement Learning Methods Applied to the Noisy Mountain Car Problem

Verena Heidrich-Meisner and Christian Igel

Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany
{Verena.Heidrich-Meisner,Christian.Igel}@neuroinformatik.rub.de

Abstract. Two variable metric reinforcement learning methods, the natural actor-critic algorithm and the covariance matrix adaptation evolution strategy, are compared on a conceptual level and analysed experimentally on the mountain car benchmark task with and without noise.

1 Introduction

Reinforcement learning (RL) algorithms address problems where an agent is to learn a behavioural policy based on reward signals, which may be unspecific, sparse, delayed, and noisy. Many different approaches to RL exist, here we consider policy gradient methods (PGMs) and evolution strategies (ESs). This paper extends our previous work on analysing the conceptual similarities and differences between PGMs and ESs [1].

For the time being, we look at single representatives of each approach that have been very successful in their respective area, the *natural actor critic* algorithm (NAC, [2,3,4,5]) and the *covariance matrix adaptation ES* (CMA-ES, [6]). Both are variable metric methods, actively learning about the structure of the search space. The CMA-ES is regarded as state-of-the-art in real-valued evolutionary optimisation [7]. It has been successfully applied and compared to other methods in the domain of RL [8,9,10,11,12]. Interestingly, recent studies compare CMA-ES and variants of the NAC algorithm in the context of optimisation [13], while we look at both methods in RL.

We promote the CMA-ES for RL because of its efficiency and, even more important, its robustness. The superior robustness compared to other RL algorithms has several reasons, but probably the most important reason is that the adaptation of the policy as well as of the metric is based on ranking policies, which is much less error prone than estimating absolute performance or performance gradients.

Our previous comparison of NAC and CMA-ES on different variants of the single pole balancing benchmark in [1] indicate that the CMA-ES is more robust w.r.t. to the choice of hyperparameters (such as initial learning rates) and initial policies compared to the NAC. In [1] the NAC performed on par with the CMA-ES in terms of learning speed only when fine-tuning policies, but worse for harder pole balancing scenarios. In this paper, we compare the two methods applied to the mountain car problem [14] to support our hypotheses and previous findings. As the considered class of policies has only two parameters, this benchmark serves as some kind of minimal working example for RL methods learning correlations between parameters. The performance of random search provides a performance baseline in our study. In order to investigate the robustness of the algorithms, we study the influence of noise added to the observations.

The paper is organised as follows. In section 2 we review the NAC algorithm and the CMA-ES for RL. Section 3 describes the conceptual relations of these two approaches and in section 4 we empirically compare the methods.

2 Reinforcement Learning Directly in Policy Space

Markov decision processes (MDP) are the basic formalism to describe RL problems. An MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ consists of the set of states $\mathcal{S}$, the possible actions $\mathcal{A}$, the probabilities $\mathcal{P}^a_{s,s'}$ that an action a taken in state s leads to state s', and the expected rewards $\mathcal{R}^a_{s,s'}$ received when going from state a s to s' after performing an action a.

Partially observable Markov decision processes (POMDP) are a generalisation of MDPs [15]. In a POMDP, the environment is determined by an MDP, but the agent cannot directly observe the state of the MPD. Formally, a POMDP can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, O \rangle$. The first four elements define the underlying MDP, Ω is the set of observations an agent can perceive, and the observation function $O : \mathcal{S} \times \mathcal{A} \rightarrow \Lambda(\Omega)$ maps a state and the action that resulted in this state to a probability distribution over observations (i.e., $O(s', a)(o)$ is the probability of observing o given that the agent took action a and landed in state s').

The goal of RL is to find a behavioral policy π such that some notion of expected future reward $\rho(\pi)$ is maximized. For example, for episodic tasks we can define $\rho(\pi) = \sum_{s,s' \in \mathcal{S}, a \in \mathcal{A}} d^\pi(s)\pi(s,a)\mathcal{P}^a_{s,s'}\mathcal{R}^a_{s,s'}$, where $d^\pi(s) = \sum_{t=0}^\infty \gamma^t \Pr\{s_t = s \mid s_0, \pi\}$ is the stationary state distribution, which we assume to exist, s_t is state in time step t, and $\gamma \in]0,1]$ a discount parameter. The immediate reward received after the action in time step t is denoted by $r_{t+1} \in \mathbb{R}$.

Most RL algorithms learn value functions measuring the quality of an action in a state and define the policy on top of these functions. Direct policy search methods and PGMs search for a good policy in a parametrised space of functions. They may build on estimated value functions (as PGMs usually do), but this is not necessary (e.g., in ESs).

2.1 Natural Policy Gradient Ascent

Policy gradient methods operate on a predefined class of stochastic policies. They require a differentiable structure to ensure the existence of the gradient of the performance measure and ascent this gradient. Let the performance $\rho(\pi)$ of the current policy with parameters $\boldsymbol{\theta}$ be defined as above. Because in general neither

d^π, $\mathcal{R}$, nor $\mathcal{P}$ are known, the performance gradient $\boldsymbol{\nabla}_{\boldsymbol{\theta}}\rho(\pi)$ with respect to the policy parameters $\boldsymbol{\theta}$ is estimated from interaction with the environment.

The policy gradient theorem [16] ensures that the performance gradient can be determined from unbiased estimates of the state-action value function $Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|\pi, s_0 = s, a_0 = a\right]$ and stationary distribution, respectively. For any MDP we have

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}}\rho = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \boldsymbol{\nabla}_{\boldsymbol{\theta}}\pi(s,a) Q^\pi(s,a) \ . \tag{1}$$

This formulation contains explicitly the unknown value function, which has to be estimated. It can be replaced by a function approximator $f_{\boldsymbol{v}} : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ (the *critic*) with real-valued parameter vector $\boldsymbol{v}$ satisfying the *convergence condition* $\sum_{s\in\mathcal{S}} d^\pi(s)\sum_{a\in\mathcal{A}}\pi(s,a)\left[Q^\pi(s,a) - f_{\boldsymbol{v}}(s,a)\right]\boldsymbol{\nabla}_{\boldsymbol{v}} f_{\boldsymbol{v}}(s,a) = 0$. This leads directly to the extension of the policy gradient theorem for function approximation. If $f_{\boldsymbol{v}}$ satisfies the convergence condition and is *compatible* with the policy parametrisation in the sense that $\boldsymbol{\nabla}_{\boldsymbol{v}} f_{\boldsymbol{v}}(s,a) = \boldsymbol{\nabla}_{\boldsymbol{\theta}}\pi(s,a)/\pi(s,a)$, that is,

$$f_{\boldsymbol{v}} = \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ln(\pi(s,a))\boldsymbol{v} + \text{const} \ , \tag{2}$$

then the policy gradient theorem holds if $Q^\pi(s,a)$ in equation 1 is replaced by $f_{\boldsymbol{v}}(s,a)$ [16].

Stochastic policies π with parameters $\boldsymbol{\theta}$ are parametrised probability distributions. In the space of probability distributions, the Fisher information matrix $F(\boldsymbol{\theta})$ induces an appropriate metric suggesting "natural" gradient ascent in the direction of $\tilde{\boldsymbol{\nabla}}_{\boldsymbol{\theta}}\rho(\pi) = F(\boldsymbol{\theta})^{-1}\boldsymbol{\nabla}_{\boldsymbol{\theta}}\rho(\pi)$. Using the definitions above, we have

$$F(\boldsymbol{\theta}) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi(s,a)\boldsymbol{\nabla}_{\boldsymbol{\theta}}\ln(\pi(s,a))(\boldsymbol{\nabla}_{\boldsymbol{\theta}}\ln(\pi(s,a)))^{\mathrm{T}} \ .$$

This implies $\boldsymbol{\nabla}_{\boldsymbol{\theta}}\rho = F(\boldsymbol{\theta})\boldsymbol{v}$, which leads to the most interesting identity

$$\tilde{\boldsymbol{\nabla}}_{\boldsymbol{\theta}}\rho(\pi) = \boldsymbol{v} \ .$$

In the following, we derive the NAC according to [4,5]. The function approximator $f_{\boldsymbol{v}}$ estimates the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, where $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|\pi, s_0 = s\right]$ is the state value function. Inserting this in the Bellman equation for Q^π leads to

$$Q^\pi(s_t,a_t) = A^\pi(s_t,a_t) + V^\pi(s_t) = \sum_{s'} P^{a_t}_{s_t,s'}\left(\mathcal{R}^{a_t}_{s_t,s'} + \gamma V^\pi(s')\right) \ . \tag{3}$$

Now we insert equation 2 for the advantage function and sum up equation 3 over a sample path:

$$\sum_{t=0}^{T}\gamma^t A^\pi(s_t,a_t) = \sum_{t=0}^{T}\gamma^t r_{t+1} + \gamma^{T+1}V^\pi(s_{T+1}) - V(s_0) \ .$$

Algorithm 1. episodic Natural Actor-Critic

```
1  initialise $\boldsymbol{\theta} = \mathbf{0} \in \mathbb{R}^n$, $\boldsymbol{\Phi} = \mathbf{0} \in \mathbb{R}^{e_{\max} \times n+1}$, $\boldsymbol{R} = \mathbf{0} \in \mathbb{R}^{e_{\max}}$
2  for $k = 1, \ldots$ do
       // $k$ counts number of policy updates
3      for $e = 1, \ldots e_{\max}$ do
           // $e$ counts number of episodes per policy update, $e_{\max} > n$
4          for $t = 1, \ldots t_{\max}$ do
               // $t$ counts number of time steps per episode
5              begin
6                  observe state $\boldsymbol{s}_t$
7                  choose action $a_t$ from $\boldsymbol{\pi_\theta}$
8                  perform action $a_t$
9                  observe reward $r_{t+1}$
10             end
11             for $i = 1, \ldots, n$ do
12                 $[\boldsymbol{\Phi}]_{e,i} \leftarrow [\boldsymbol{\Phi}]_{e,i} + \gamma^t \frac{\partial}{\partial \theta_i} \ln \boldsymbol{\pi_\theta}(s_t, a_t)$
13             $[\boldsymbol{R}]_e \leftarrow [\boldsymbol{R}]_e + \gamma^t r_{t+1}$
14         $[\boldsymbol{\Phi}]_{e,n+1} \leftarrow 1$
       // update policy parameters:
15     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + (\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{R}$
```

For an episodic task terminating in time step T it holds $V^\pi(s_{T+1}) = 0$. Thus, we have after replacing A^π with its approximation according to equation 2:

$$\sum_{t=0}^{T} \gamma^t (\boldsymbol{\nabla_\theta} \ln \pi(s_t, a_t))^{\mathrm{T}} \boldsymbol{v} - V(s_0) = \sum_{t=0}^{T} \gamma^t r_{t+1}$$

For fixed start states we have $V^\pi(s_0) = \rho(\pi)$ and this is a linear regression problem with $n+1$ unknown variables $\boldsymbol{w} = [\boldsymbol{v}^{\mathrm{T}}, V^\pi(s_0)]^{\mathrm{T}}$ that can be solved after $n+1$ observed episodes (where n is the dimension of $\boldsymbol{\theta}$ and $\boldsymbol{v}$):

$$\begin{array}{ccc}
\left[\sum_{t=0}^{T(e_1)} \left[\gamma^t \boldsymbol{\nabla_\theta} \ln \pi(s_t^{e_1}, a_t^{e_1}) \right]^{\mathrm{T}}, -1 \right]^{\mathrm{T}} \boldsymbol{v} & = & \sum_{t=0}^{T(e_1)} \gamma^t r_{t+1}^{e_1} \\
\vdots & & \vdots \\
\left[\sum_{t=0}^{T(e_n)} \left[\gamma^t \boldsymbol{\nabla_\theta} \ln \pi(s_t^{e_n}, a_t^{e_n}) \right]^{\mathrm{T}}, -1 \right]^{\mathrm{T}} \boldsymbol{v} & = & \sum_{t=0}^{T(e_n)} \gamma^t r_{t+1}^{e_n}
\end{array}$$

The superscripts indicate the episodes. In algorithm 1 the likelihood information for a sufficient number of episodes is collected in a matrix $\boldsymbol{\Phi}$ and the return for each episode in $\boldsymbol{R}$. In every update step one inversion of the matrix $\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}$ is necessary.

2.2 Covariance Matrix Adaptation Evolution Strategy

We consider ESs for real-valued optimisation [17,18,19,20]. Let the optimisation problem be defined by an objective function $f : \mathbb{R}^n \to \mathbb{R}$ to be minimised, where n denotes the dimensionality of the search space (space of candidate solutions, decision space). Evolution strategies are random search methods, which iteratively sample a set of candidate solutions from a probability distribution over the search space, evaluate these points using f, and construct a new probability distribution over the search space based on the gathered information. In ESs, this search distribution is parametrised by a set of candidate solutions, the *parent population* with size μ, and by parameters of the variation operators that are used to create new candidate solutions (the *offspring population* with size λ) from the parent population.

In each iteration k, the lth offspring $\boldsymbol{x}_l \in \mathbb{R}^n$ is generated by multi-variate *Gaussian mutation* and *weighted global intermediate recombination*, i.e.,

$$\boldsymbol{x}_l^{(k+1)} = \left\langle \boldsymbol{x}_{\text{parents}}^{(k)} \right\rangle_{\boldsymbol{w}} + \sigma^{(k)} \boldsymbol{z}_l^{(k)} \ ,$$

where $\boldsymbol{z}_l^{(k)} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{C}^{(k)})$ and $\left\langle \boldsymbol{x}_{\text{parents}}^{(k)} \right\rangle_{\boldsymbol{w}} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i\text{th-best-parent}}^{(k)}$ (a common choice is $\boldsymbol{w}_i \propto \ln(\mu+1) - \ln(i)$, $\|\boldsymbol{w}\|_1 = 1$).

The CMA-ES, shown in algorithm 2, is a variable metric algorithm adapting both the n-dimensional covariance matrix $\boldsymbol{C}^{(k)}$ of the normal mutation distribution as well as the *global step size* $\sigma^{(k)} \in \mathbb{R}^+$. In the basic algorithm, a low-pass filtered *evolution path* $\boldsymbol{p}^{(k)}$ of successful (i.e., selected) steps is stored,

$$\boldsymbol{p}_c^{(k+1)} \leftarrow (1 - c_c)\, \boldsymbol{p}_c^{(k)} + \sqrt{(c_c(2 - c_c)\mu_{\text{eff}})}\, \frac{1}{\sigma^{(k)}} \left(\left\langle \boldsymbol{x}_{\text{parents}}^{(k+1)} \right\rangle - \left\langle \boldsymbol{x}_{\text{parents}}^{(k)} \right\rangle \right) ,$$

and $\boldsymbol{C}^{(k)}$ is changed to make steps in the promising direction $\boldsymbol{p}^{(k+1)}$ more likely:

$$\boldsymbol{C}^{(k+1)} \leftarrow (1 - c_{\text{cov}})\, \boldsymbol{C}^{(k)} + c_{\text{cov}}\, \boldsymbol{p}_c^{(k+1)} \boldsymbol{p}_c^{(k+1)^{\mathrm{T}}}$$

(this rank-one update of $\boldsymbol{C}^{(k)}$ can be augmented by a rank-μ update, see [21]). The variables c_c and c_{cov} denote fixed learning rates. The learning rate $c_{\text{cov}} = \frac{2}{(n+\sqrt{2})^2}$ is roughly inversely proportional to the degrees of freedom of the covariance matrix. The backward time horizon of the cumulation process is approximately c_c^{-1}, with $c_c = 4/(n+4)$ linear in the dimension of the path vector. Too small values for c_c would require an undesirable reduction of the learning rate for the covariance matrix. The *variance effective selection mass* $\mu_{\text{eff}} = \left(\sum_{t=1}^{\mu} w_i^2\right)^{-1}$ is a normalisation constant.

The global step size $\sigma^{(k)}$ is adapted on a faster timescale. It is increased if the selected steps are larger and/or more correlated than expected and decreased if they are smaller and/or more anticorrelated than expected:

$$\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{E\|\mathrm{N}(\boldsymbol{0}, \boldsymbol{I})\|} - 1 \right) \right) \ ,$$

Algorithm 2. rank-one CMA-ES

```
1  initialise $\boldsymbol{m}^{(0)} = \boldsymbol{\theta}$ and $\sigma^{(0)}$, evolution path $\boldsymbol{p}_\sigma^{(0)} = 0, \boldsymbol{p}_c^{(0)} = 0$ and covariance
   matrix $\boldsymbol{C}^{(0)} = \boldsymbol{I}$ (unity matrix)
2  for $k = 1, \ldots$ do
       // $k$ counts number generations respective of policy updates
3      for $l = 1, \ldots, \lambda$ do
4          $x_l^{(k+1)} \sim \mathrm{N}(\boldsymbol{m}^{(k)}, {\sigma^{(k)}}^2 \boldsymbol{C}^{(k)})$ // create new offspring
       // evaluate offspring:
5      for $l = 1, \ldots, \lambda$ do
6          $f_l \leftarrow 0$ // fitness of $l$th offspring
7          for $e = 1, \ldots e_{max}$ do
               // counts number of episodes per policy update
8              for $t = 1, \ldots t_{max}$ do
                   // $t$ counts number of time steps per episode
9                  begin
10                     observe state $\boldsymbol{s}_t$,
11                     choose action $a_t$ from $\pi_\theta$,
12                     perform action $a_t$,
13                     observe reward $r_{t+1}$
14                 end
15                 $f_l \leftarrow f_l + \gamma^{t-1} r_{t+1}$
       // selection and recombination:
16     $\boldsymbol{m}^{(k+1)} \leftarrow \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(k)}$
       // step size control:
17     $\boldsymbol{p}_\sigma^{(k+1)} \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\mathrm{eff}}} {\boldsymbol{C}^{(k)}}^{-\frac{1}{2}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$
18     $\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp \frac{c_\sigma}{d_\sigma} \left( \frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{E\|\mathrm{N}(\boldsymbol{0},\boldsymbol{I})\|} - 1 \right)$
       // covariance matrix update:
19     $\boldsymbol{p}_c^{(k+1)} \leftarrow (1 - c_c)\boldsymbol{p}_c^{(k)} + \sqrt{c_c(2 - c_c)\mu_{\mathrm{eff}}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$
20     $\boldsymbol{C}^{(k+1)} \leftarrow (1 - c_{\mathrm{cov}})\boldsymbol{C}^{(k)} + c_{\mathrm{cov}} \boldsymbol{p}_c^{(k+1)} {\boldsymbol{p}_c^{(k+1)}}^{\mathrm{T}}$
```

and its (*conjugate*) evolutions path is:

$$\boldsymbol{p}_s^{(k+1)} \leftarrow (1 - c_\sigma)\, \boldsymbol{p}_s^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\mathrm{eff}}}\, {\boldsymbol{C}^{(k)}}^{-\frac{1}{2}} \left(\left\langle \boldsymbol{x}_{\mathrm{parents}}^{(k+1)} \right\rangle - \left\langle \boldsymbol{x}_{\mathrm{parents}}^{(k)} \right\rangle \right)$$

Again, $c_\sigma = \frac{\mu_{\mathrm{eff}}+2}{n+\mu_{\mathrm{eff}}+3}$ is a fixed learning rate and $d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{\mathrm{eff}}-1}{n+1}}\right) + c_\sigma$ a damping factor. The matrix $\boldsymbol{C}^{-\frac{1}{2}}$ is defined as $\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^{\mathrm{T}}$, where $\boldsymbol{B}\boldsymbol{D}^2\boldsymbol{B}^{\mathrm{T}}$ is an eigendecomposition of $\boldsymbol{C}$ ($\boldsymbol{B}$ is an orthogonal matrix with the eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{D}$ a diagonal matrix with the corresponding eigenvalues) and sampling $\mathrm{N}(\boldsymbol{0}, \boldsymbol{C})$ is done by sampling $\boldsymbol{B}\boldsymbol{D}\mathrm{N}(\boldsymbol{0}, \boldsymbol{I})$.

The values of the learning rates and the damping factor are well considered and have been validated by experiments on many basic test functions [21]. *They need not be adjusted dependent on the problem and are therefore* no *hyperparameters of the algorithm.* Also the population sizes can be set to default values, which are $\lambda = \max(4 + \lfloor 3 \ln n \rfloor, 5)$ and $\mu = \lfloor \frac{\lambda}{2} \rfloor$ for offspring and parent population, respectively [21]. If we fix $\boldsymbol{C}^{(0)} = \boldsymbol{I}$, the only (also adaptive) hyperparameter that has to be chosen problem dependent is the initial global step size $\sigma^{(0)}$.

The CMA-ES uses rank-based selection. The best μ of the λ offspring form the next parent population.

The highly efficient use of information and the fast adaptation of σ and $\boldsymbol{C}$ makes the CMA-ES one of the best direct search algorithms for real-valued optimisation [7]. For a detailed description of the CMA-ES see the articles by Hansen et al. [6,21,22,23].

3 Similarities and Differences of NAC and CMA-ES

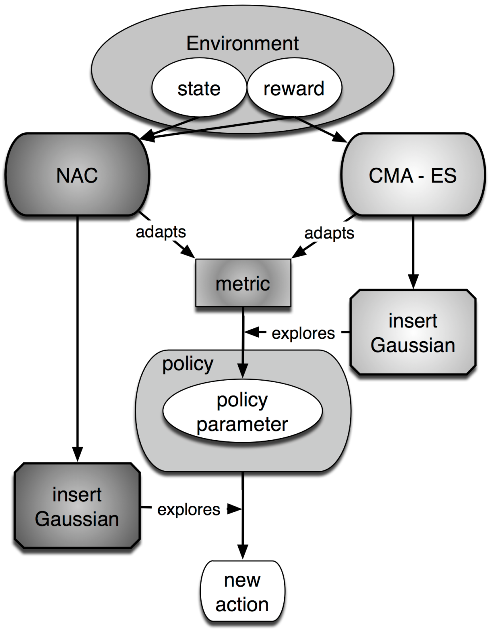


Fig. 1. Conceptual similarities and differences of natural policy gradient ascent and CMA evolution strategy: Both methods adapt a metric for the variation of the policy parameters based on information received from the environment. Both explore by stochastic perturbation of policies, but at different levels.

Policy gradient methods and ESs share several constituting aspects, see Fig. 1. Both search directly in policy space, thus the actor-part in the agent is represented and learnt actively. Yet, while ESs are actor-only methods, the NAC has an actor-critic architecture. In both approaches the class of possible policies is given by a parametrised family of functions, but in the case of PGMs the choice of the policy class is restricted to differentiable functions.

Exploration of the search space is realised by random perturbations in both ESs and PGMs. Evolutionary methods usually perturb a deterministic policy by mutation and recombination, while in PGMs the random variations are an inherent property of the stochastic policies. In ESs there is only one initial stochastic variation per policy update. In contrast, the stochastic policy introduces perturbations in every step of the episode. While the number n of parameters of the policy determines the n-dimensional random variation in the CMA-ES, in the PGMs the usually lower dimensionality of the action corresponds to the dimensionality of the random perturbations. In ESs the search is driven solely by ranking policies and not by the exact values of performance estimates or their gradients. The reduced number of random events and the rank-based evaluation are decisive differences and we hypothesise that they allow ESs to be more robust.

The CMA-ES as well as the NAC are variable-metric methods. A natural policy gradient method implicitly estimates the Fisher metric to follow the natural gradient of the performance in the space of the policy parameters and chooses its action according to a stochastic policy. Assuming a Gaussian distribution of the actions this resembles the CMA-ES. In the CMA-ES the parameters are perturbed according to a multi-variate Gaussian distribution. The covariance matrix of this distribution is adapted online. This corresponds to learning an appropriate metric for the optimisation problem at hand. After the stochastic variation the actions are chosen deterministically.

Thus, both types of algorithms perform the same conceptual steps to obtain the solution. They differ in the order of these steps and the level at which the random changes are applied.

Policy gradient methods have the common properties of gradient techniques. They are powerful local search methods and thus benefit from a starting point close to an optimum. However, they are susceptible to being trapped in undesired local minima.

4 Experiments

The experiments conducted in this paper extend our previous work described in [1]. We have chosen the mountain car problem, which is a well-known benchmark problem in RL requiring few policy parameters.

The objective of this task is to navigate an underpowered car from a valley to a hilltop. The state $\boldsymbol{s}$ of the system is given by the position $x \in [-1.2, 0.6]$ of the car and by its current velocity $v = \dot{x} \in [-0.07, 0.07]$, actions are discrete forces applied to the car $a \in \{-a_{\max}, 0, a_{\max}\}$, where $a_{\max}$ is chosen to be insufficient to drive the car directly uphill from the starting position in the valley to the goal at the top. The agent receives a negative reward of $r = -1$ for every time step. An episode terminates when the car reaches the position $x = 0.5$, the discount parameter is set to $\gamma = 1$.

To allow for a fair comparison, both methods operate on the same policy class $\pi_{\boldsymbol{\theta}}^{\text{deter}}(\boldsymbol{s}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{s}$ with $\boldsymbol{s}, \boldsymbol{\theta} \in \mathbb{R}^2$ The continuous output a_{cont} of the policy is mapped by the environment to a discrete action $a \in \mathcal{A}$: $a = 1$ if $a_{\text{cont}} > 0.1$, $a = -1$ if $a_{\text{cont}} < 0.1$, and $a = 0$ otherwise. We also considered the mountain car problem with continuous actions, which, however, makes the task easier for the CMA-ES and more difficult for the NAC. For learning, the NAC uses the stochastic policy $\pi_{\boldsymbol{\theta}}^{\text{stoch}}(\boldsymbol{s}, a) = \mathrm{N}(\pi_{\theta}^{\text{deter}}(\boldsymbol{s}), \sigma_{\text{NAC}})$, where the variance σ_{NAC} is viewed as an additional adaptive parameter of the PGM. The NAC is evaluated on the corresponding deterministic policy. In all experiments the same number of $e_{\max} = 10$ episodes is used for assessing the performance of a policy. We analyse two sets of start policies: $\boldsymbol{\theta} = \mathbf{0}$ (referred to as P_0) and drawing the components of $\boldsymbol{\theta}$ uniformly from $[-100., 100]$ (termed P_{100}). P_0 lies reasonably close to the optimal parameter values. In the original mountain car task the start states for each episode are drawn randomly from the complete state space $\mathcal{S}$ (S_{random}). We additionally analyse the case (S_{fixed}) where all episodes start in the same state with position $x = -0.8$ and velocity $v = 0.01$. Driving simply in the direction of the goal is not sufficient to solve the problem for this starting condition.

As a baseline comparison we considered stochastic search, where policy parameters were drawn uniformly at random from a fixed interval and were then evaluated in the same way as CMA-ES and NAC.

In a second set of experiments we add Gaussian noise with zero mean and variance $\sigma_{\text{noise}} = 0.01$ to state observations (i.e., now we consider a POMDP).

Mountain car task without noise. Figure 2 shows the performance of NAC and CMA-ES on the mountain car problem. In the easiest cases (P_0 with S_{random} and S_{fixed}) the NAC clearly outperforms the CMA-ES. Here the NAC is also

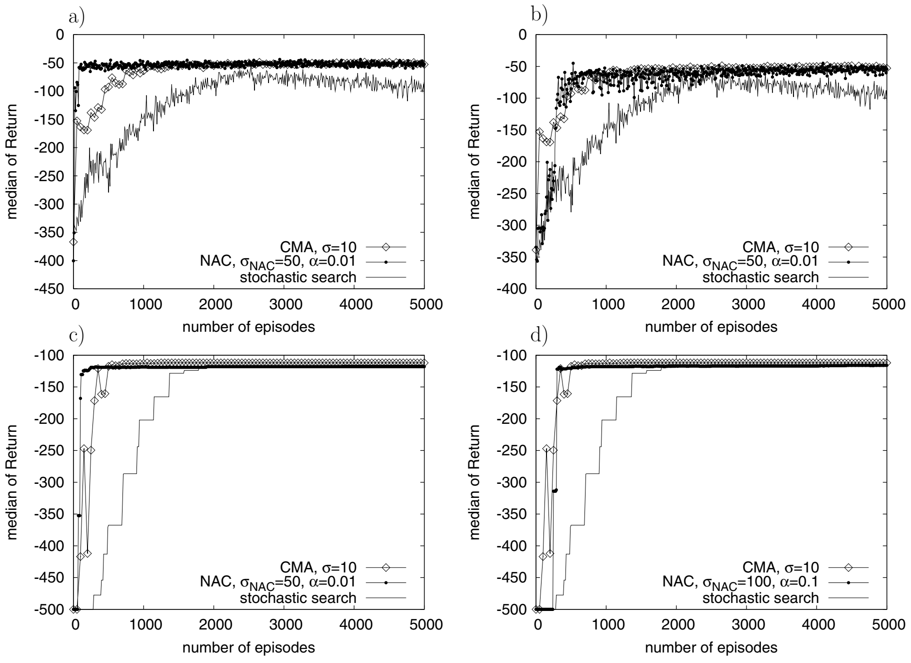


Fig. 2. Performance of NAC and CMA-ES on the mountain car task without noise based on 20 trials. a) CMA-ES, NAC, and stochastic search for initial policy P_0 and initial environment state S_{random} (with best respective parameter values) without noise b) CMA-ES, NAC, and stochastic search for initial policy P_{100} and initial environment state S_{random} (with best respective parameter values) without noise c) CMA-ES, NAC, and stochastic search for initial policy P_0 and initial environment state S_{fixed} (with best respective parameter values) without noise d) CMA-ES, NAC, and stochastic search for initial policy P_{100} and initial environment state S_{fixed} (with best respective parameter values) without noise.

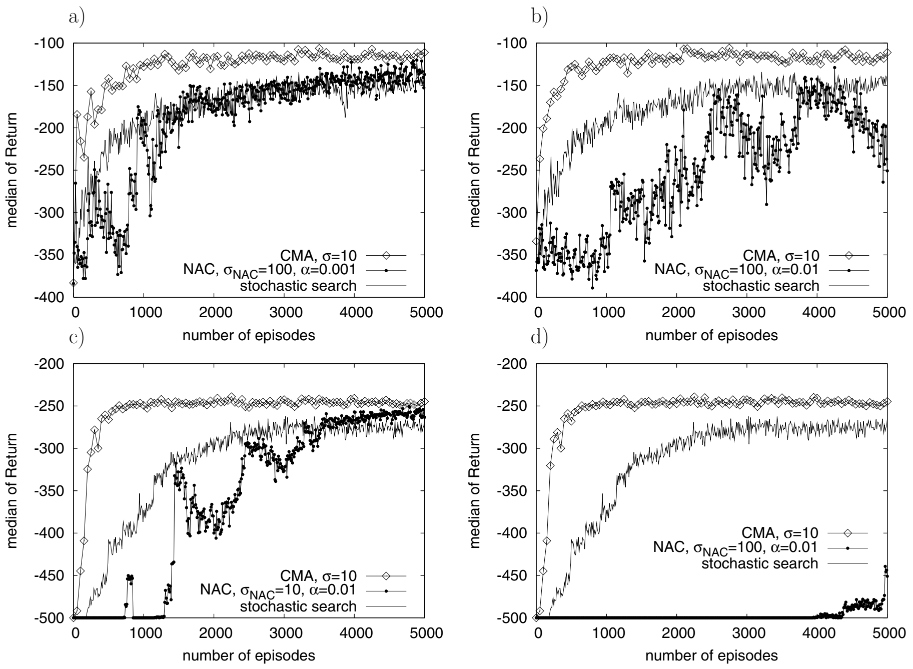


Fig. 3. Performance of NAC and CMA-ES on the mountain car task with noisy observations based on 20 trials. a) CMA-ES, NAC, and stochastic search for initial policy P_0 and initial environment state S_{random} (with best respective parameter values) with noise b) CMA-ES, NAC, and stochastic search for initial policy P_{100} and initial environment state S_{random} (with best respective parameter values) with noise c) CMA-ES, NAC, and stochastic search for initial policy P_0 and initial environment state S_{fixed} (with best respective parameter values) with noise d) CMA-ES, NAC, and stochastic search for initial policy P_{100} and initial environment state S_{fixed} (with best respective parameter values) with noise.

robust w.r.t changes of its hyperparameters (learning rate and initial variance). But this changes when the policy is not initialised close to the optimal parameter values and the parameters are instead drawn randomly. The CMA-ES performs as in the former case, but now it is faster than the NAC in the beginning. The NAC still reaches an optimal solution faster, but it is no longer robust. The CMA-ES is more stable in all cases. Its performance does not depend on the choice of initial policy at all and changing the value of its single parameter σ as the initial step size only marginally effects the performance, see figure 4 and tables 1 and 2.

Mountain car task with noise. For the next set of experiments we added noise to the observed state, thus creating a more realistic situation, see figure 3. In this case the CMA-ES clearly outperforms the NAC while still being robust with respect to the initialisation of the policy parameters and the choice of the initial

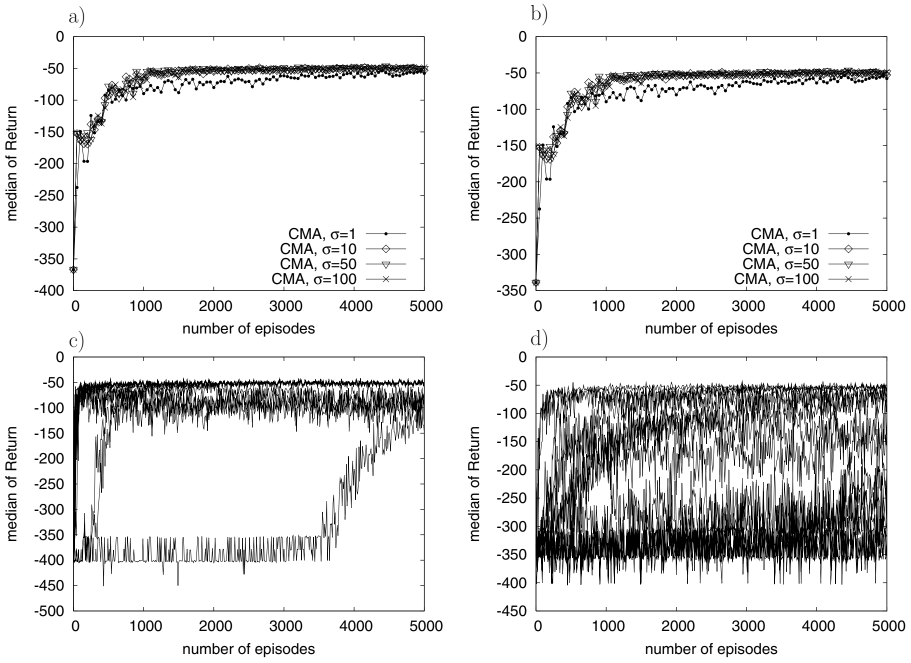


Fig. 4. Robustness against changes in the respective parameters on the mountain car task without noise. To avoid overcrowding plots only the worst-case examples are shown here: a) CMA-ES for initial policy P_0 and initial environment state S_{random} without noise and $\sigma \in [1, 10, 50, 100]$. b) CMA-ES for initial policy P_{100} and initial environment state S_{random} without noise and $\sigma \in [1, 10, 50, 100]$. c) NAC for initial policy P_0 and initial environment state S_{random} without noise, ordered from top to bottom by their final value as given in table 1, d) NAC for initial policy P_{100} and initial environment state S_{random} without noise, ordered from top to bottom by their final value as given in table 2.

Table 1. Final performance values of NAC on the mountain car task without noise with initial policy P_0 and starting states from S_{random} after 5000 episodes. The medians of 20 trials are reported.

α	0.0001	0.001	0.0001	0.0001	0.001	0.01	0.01	0.001
σ_{NAC}	10	10	100	50	100	1	10	1
final value	−49.4	−49.4	−49.55	−49.55	−50.7	−50.75	−50.85	−51.4
α	0.001	0.01	0.01	0.1	0.1	0.1	0.1	0.0001
σ_{NAC}	50	50	100	10	100	1	50	1
final value	−51.45	−52.35	−53.6	−73.3	−82.2	−98.55	−131.6	−147.45

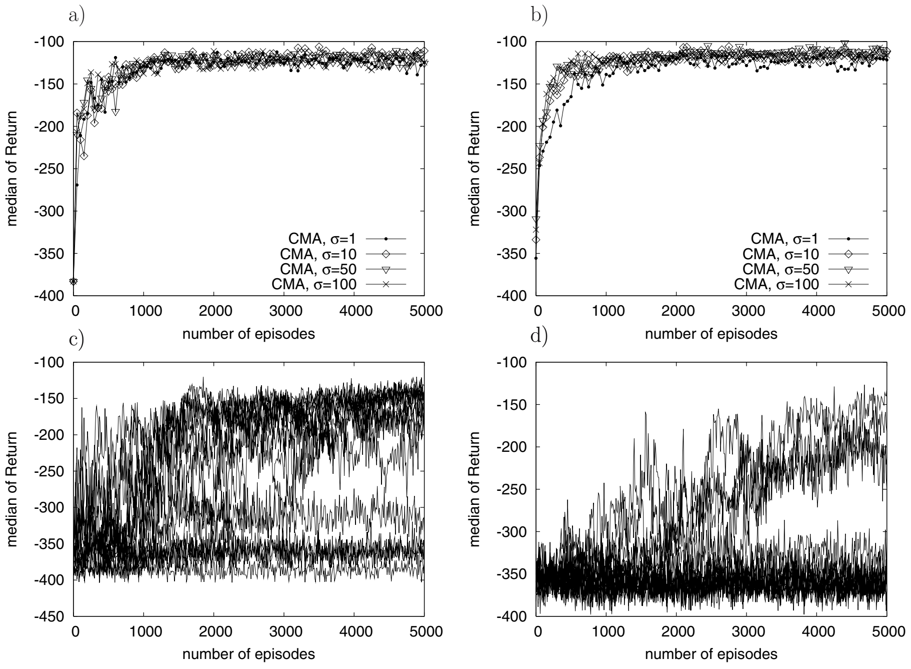


Fig. 5. Robustness against changes in the respective parameters on the mountain car task with noise. Again in order to avoid overcrowding plots only the worst-case examples are shown: a) CMA-ES for initial policy P_0 and initial environment state S_{random} with noise and $\sigma \in [1, 10, 50, 100]$. b) CMA-ES for initial policy P_{100} and initial environment state S_{random} with noise and $\sigma \in [1, 10, 50, 100]$. c) NAC for initial policy P_0 and initial environment state S_{random} with noise, ordered from top to bottom by their final value as given in table 3, d) NAC for initial policy P_{100} and initial environment state S_{random} with noise, ordered from top to bottom by their final value as given in table 4.

Table 2. Final performance values of NAC on the mountain car task without noise with initial policy P_{100} and starting states from S_{random} after 5000 epsisodes

α	0.001	0.01	0.01	0.001	0.1	0.01	0.1	0.1
σ_{NAC}	100	50	100	50	50	10	100	10
final value	-53.2	-54.25	-54.4	-59.15	-70.7	-84.45	-113.1	-117.25

α	0.0001	0.01	0.1	0.0001	0.0001	0.001	0.001	0.0001
σ_{NAC}	100	1	1	50	10	10	1	1
final value	-200.4	-289.15	-307.35	-309.2	-314.85	-316.1	-352.05	-354

Table 3. Final performance values of NAC on the mountain car task with noisy observations with initial policy P_0 and starting states from S_{random} after 5000 epsisodes

α	0.001	0.001	0.01	0.0001	0.001	0.001	0.01	0.01
σ_{NAC}	50	100	100	100	10	1	1	50
final value	−133.17	−137.16	−137.83	−144.56	−164.71	−171.69	−174.34	−178.76
α	0.01	0.0001	0.0001	0.1	0.1	0.0001	0.1	0.1
σ_{NAC}	10	10	50	10	50	1	100	1
final value	−184.03	−185.15	−201.84	−336.33	−353.91	−377.36	−377.66	−380.71

Table 4. Final performance values of NAC on the mountain car task with noisy observations with initial policy P_{100} and starting states from S_{random} after 5000 epsisodes

α	0.01	0.001	0.001	0.01	0.1	0.0001	0.01	0.0001
σ_{NAC}	100	100	50	50	50	1	1	100
final value	−131.81	−182.24	−212.23	−250.83	−308.37	−343.58	−349.77	−350.2
α	0.01	0.0001	0.001	0.1	0.1	0.0001	0.01	0.1
σ_{NAC}	10	50	1	10	100	10	10	1
final value	−351.04	−358.48	−362.8	−367.38	−368.65	−368.83	−371.67	−378.16

step size, see figure 5 and tables 3 and 4. NAC performs at best on par with stochastic search, for the more difficult policy initialisation P_{100} it is even worse.

5 Conclusion

The covariance matrix adaptation evolution strategy (CMA-ES) applied to reinforcement learning (RL) is conceptually similar to policy gradient methods with variable metric such as the natural actor critic (NAC) algorithm. However, we argue that the CMA-ES is much more robust w.r.t. the choice of hyperparameters, policy initialisation, and especially noise. On the other hand, given appropriate hyperparameters, the NAC can outperform the CMA-ES in terms of learning speed if initialised close to a desired policy. The experiments in this paper on the noisy mountain car problem and our previous results on the pole balancing benchmark support these conjectures. Across the different scenarios, the CMA-ES proved to be a highly efficient direct RL algorithm. The reasons for the robustness of the CMA-ES are the powerful adaptation mechanisms for the search distribution and the rank-based evaluation of policies.

In future work we will extend the experiments to different and more complex benchmark tasks and to other direct policy search methods.

Acknowledgement. The authors acknowledge support from the German Federal Ministry of Education and Research within the Bernstein group "The grounding of higher brain function in dynamic neural fields".

References

1. Heidrich-Meisner, V., Igel, C.: Similarities and differences between policy gradient methods and evolution strategies. In: Verleysen, M. (ed.) 16th European Symposium on Artificial Neural Networks (ESANN), Evere, Belgium, pp. 149–154. d-side publications (2008)
2. Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: Proc. 3rd IEEE-RAS Int'l. Conf. on Humanoid Robots, pp. 29–30 (2003)
3. Riedmiller, M., Peters, J., Schaal, S.: Evaluation of policy gradient methods and variants on the cart-pole benchmark. In: Proc. 2007 IEEE Internatinal Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007), pp. 254–261 (2007)
4. Peters, J., Schaal, S.: Applying the episodic natural actor-critic architecture to motor primitive learning. In: Proc. 15th European Symposium on Artificial Neural Networks (ESANN 2007), Evere, Belgium, pp. 1–6. d-side publications (2007)
5. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing 71(7-9), 1180–1190 (2008)
6. Hansen, N.: The CMA evolution strategy: A comparing review. In: Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pp. 75–102. Springer, Heidelberg (2006)
7. Beyer, H.G.: Evolution strategies. Scholarpedia 2(18), 1965 (2007)
8. Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: Congress on Evolutionary Computation (CEC 2003), vol. 4, pp. 2588–2595. IEEE Press, Los Alamitos (2003)
9. Pellecchia, A., Igel, C., Edelbrunner, J., Schöner, G.: Making driver modeling attractive. IEEE Intelligent Systems 20(2), 8–12 (2005)
10. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Efficient non-linear control through neuroevolution. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS, vol. 4212, pp. 654–662. Springer, Heidelberg (2006)
11. Siebel, N.T., Sommer, G.: Evolutionary reinforcement learning of artificial neural networks. International Journal of Hybrid Intelligent Systems 4(3), 171–183 (2007)
12. Kassahun, Y., Sommer, G.: Efficient reinforcement learning through evolutionary acquisition of neural topologies. In: Verleysen, M. (ed.) 13th European Symposium on Artificial Neural Networks, pp. 259–266. d-side (2005)
13. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: Computational Intelligence: Research Frontiers. IEEE Press, Los Alamitos (accepted, 2008)
14. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
15. Kaelbling, L., Littman, M., Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101(1-2), 99–134 (1998)
16. Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Advances in Neural Information Processing Systems 12, 1057–1063 (2000)
17. Rechenberg, I.: Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog (1973)
18. Schwefel, H.P.: Evolution and Optimum Seeking. Sixth-Generation Computer Technology Series. John Wiley & Sons, Chichester (1995)

19. Beyer, H.G., Schwefel, H.P.: Evolution strategies: A comprehensive introduction. Natural Computing 1(1), 3–52 (2002)
20. Kern, S., Müller, S., Hansen, N., Büche, D., Ocenasek, J., Koumoutsakos, P.: Learning probability distributions in continuous evolutionary algorithms – A comparative review. Natural Computing 3, 77–112 (2004)
21. Hansen, N., Müller, S., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
22. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
23. Hansen, N., Niederberger, A.S.P., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Transactions on Evolutionary Computation (in press, 2008)

Optimistic Planning of Deterministic Systems

Jean-François Hren and Rémi Munos

SequeL project, INRIA Lille - Nord Europe
40 avenue Halley, 59650 Villeneuve d'Ascq, France
{jean-francois.hren,remi.munos}@inria.fr

Abstract. If one possesses a model of a controlled deterministic system, then from any state, one may consider the set of all possible reachable states starting from that state and using any sequence of actions. This forms a tree whose size is exponential in the planning time horizon. Here we ask the question: given finite computational resources (e.g. CPU time), which may not be known ahead of time, what is the best way to explore this tree, such that once all resources have been used, the algorithm would be able to propose an action (or a sequence of actions) whose performance is as close as possible to optimality? The performance with respect to optimality is assessed in terms of the regret (with respect to the sum of discounted future rewards) resulting from choosing the action returned by the algorithm instead of an optimal action. In this paper we investigate an optimistic exploration of the tree, where the most promising states are explored first, and compare this approach to a naive uniform exploration. Bounds on the regret are derived both for uniform and optimistic exploration strategies. Numerical simulations illustrate the benefit of optimistic planning.

1 Introduction

This paper is concerned with the problem of making the best possible use of available numerical resources (such as CPU time, memory, number of calls to a black-box model, ...) in order to solve a sequential decision making problem in deterministic domains. We aim at designing *anytime algorithms*, i.e. which return higher accuracy solutions whenever additional resources are provided. To fix the setting, we are interested in generating a near-optimal policy for a deterministic system with discounted rewards, under finite action space but large state space (possibly infinite). We assume that we have a generative model of the state dynamics and rewards.

The approach consists in considering at each time-step t, the look-ahead tree (which may be constructed from our model) of all possible reachable states when using any sequence of actions, starting from x_t. Then a search is performed in this tree by using a specific exploration strategy. We address the question of how should one explore this tree such that after a finite number of numerical resources (here we will consider the number of node expansions or node transitions, which is directly related to the CPU time, or the number of calls to our model), we would be able to make the best possible decision about the action (or sequence of actions) to choose from state x_t. This action (or sequence) is subsequently executed in the real world, and the overall process is repeated from the next state x_{t+1}. This is close in spirit to the sparse sampling algorithm of [KMN02], where a sampling device is used to explore the look-ahead tree and generate a near optimal action with high probability. Their setting is more general than ours since they consider general Markov Decision Processes, whereas we restrict ourself to deterministic dynamics and rewards (we will consider the stochastic case in future work). However the purpose of our study is the possible clever exploration of the tree, whereas they only consider uniform exploration. Empirical works, such as [PG04], suggest that non-uniform exploration may greatly improve the performance of the resulting policy, given a fixed amount of computation.

The bounds on the performance we obtain here do not depend on the dimension of the state space, contrarily to usual (possibly approximate) Dynamic Programming and Reinforcement Learning approaches (where a close-to-optimal policy results from the approximation of a value function over the whole domain), see [Put94, BT96, SB98], which are subject to the curse of dimensionality. However our bounds scale with the branching factor of some related trees and the time horizon $1/(\log 1/\gamma)$ (where γ is the discount factor). The performance measure we consider here is the regret $R(n)$, which is the performance loss (with respect to optimality) of the decision returned by the algorithm after having explored the tree using n units of computational resources. This notion will be made precise in the next section. If we consider a uniform exploration of the tree, one obtains the (upper and lower) bounds on the regret: $R(n) = \Theta(n^{-\frac{\log 1/\gamma}{\log K}})$, where K is the number of actions (the branching factor of the full look-ahead tree). We thus expect such approaches to be interesting (compared to value function-based approaches) when the state space is huge but the number of actions is relatively small.

In this paper we investigate an **optimistic exploration** of the tree, where the most promising nodes are explored first. The idea is to explore at each round the node that has a possibility of being the best (which motivates the term "optimistic"), in the sense of having the highest upper-bound. The idea of selecting actions based on upper confidence bounds (UCB) dates to early work in multi-armed bandit problems, see e.g. [Rob52, LR85], and more recently [ACBF02]. Planning under uncertainty using UCB has been considered in [KS06] and the resulting algorithm UCT (UCB applied to Trees) has been successfully applied to the large scale tree search problem of computer-go, see [GWMT06]. However, regret analysis shows that UCT may perform very poorly because of overly-optimistic assumptions of the bounds, see [CM07]. Our work is close in spirit to the BAST (Bandit Algorithm for Smooth Trees) algorithm of [CM07], where we use the inherent smoothness of the look-ahead tree (which comes from the fact that we consider the sum of discounted rewards along the paths) to settle true upper-bounds on the nodes value. Using this optimistic exploration, our main result is an upper-bound on the regret $R(n) = O(n^{-\frac{\log 1/\gamma}{\log \kappa}})$, where $\kappa \in (1, K]$ is the branching factor of a related subtree, composed of all the nodes that will eventually have to be evaluated in order to decide whether they belong to an optimal path or not. In particular, the optimistic exploration is never worst than the uniform one (in some sense made precise later), and much better whenever κ is smaller than K and close to 1. We show that κ is also related to the proportion of near-optimal paths in the full look-ahead tree, and that in hard instances of search problems, κ is small compared to K, which increases the benefit of using optimistic rather than uniform exploration for complex problems. We further show that in some non-trivial cases, $\kappa = 1$, and exponential rates are derived.

In the next section, we introduce the notations and motivations for the tree exploration problem. Then we consider the uniform and optimistic exploration strategies. We conclude the paper by numerical experiments illustrating the benefit of the optimistic exploration.

2 Planning Under Finite Numerical Resources

We are interested in making the best possible use of available numerical resources for decision making. For that purpose, we assume that we possess a generative model that can be used to generate simulated transitions and rewards. The action-selection procedure (call it $\mathcal{A}$) takes as input the current state of the system and outputs an action $\mathcal{A}(n)$ (or a sequence of actions, but we will focus in this paper on the single output action case) using a finite number n of available numerical resources. The term resource refers to a piece of computational effort which may be measured e.g. in terms of CPU time or number of calls to the generative model. The amount n of available resources may not be known before they are all used, so we wish to design anytime algorithms. Our goal is that the proposed action $\mathcal{A}(n)$ be as close as possible to the optimal action in that state, in the sense that the regret $R_{\mathcal{A}}(n)$ (in the cumulative discounted sum of rewards to come) of choosing this action instead of the optimal one should be small. Let us now introduce some notations to define more precisely this notion of regret.

We consider here a deterministic controlled problem defined by the tuple (X, A, f, r), where X is the state space, A the action space, $f : X \times A \to X$ the transition dynamics, and $r : X \times A \to \mathbb{R}$ the reward function. If at time t, the system is in state $x_t \in X$ and the action a_t is chosen, then the system jumps to the next state $x_{t+1} = f(x_t, a_t)$ and a reward $r(x_t, a_t)$ is received. In this paper we will assume that all rewards are in the interval $[0, 1]$. We assume that the state space is large (possibly infinite), and the action space is finite, with K possible actions. We consider an infinite time horizon problem with discounted rewards ($0 \leq \gamma < 1$ is the discount factor). For any policy $\pi : X \to A$ we define the value function $V^\pi : X \to \mathbb{R}$ associated to that policy:

$$V^\pi(x) \overset{\text{def}}{=} \sum_{t \geq 0} \gamma^t r(x_t, \pi(x_t)),$$

where x_t is the state of the system at time t when starting from x (i.e. $x_0 = x$) and following policy π.

We also define the Q-value function $Q^\pi : X \times A \to \mathbb{R}$ associated to a policy π, in state-action (x, a), as:

$$Q^\pi(x,a) \overset{\text{def}}{=} r(x,a) + \gamma V^\pi(f(x,a)).$$

We have the property that $V^\pi(x) = Q^\pi(x, \pi(x))$. Now the optimal value function (respectively Q-value function) is defined as: $V^*(x) \overset{\text{def}}{=} \sup_\pi V^\pi(x)$ (respectively $Q^*(x,a) \overset{\text{def}}{=} \sup_\pi Q^\pi(x,a)$). From the dynamic programming principle, we have the Bellman equations:

$$V^*(x) = \max_{a \in A} \big[r(x,a) + \gamma V^*(f(x,a)) \big]$$
$$Q^*(x,a) = r(x,a) + \gamma \max_{b \in A} Q^*(f(x,a), b).$$

Now, let us return to the action-selection algorithm $\mathcal{A}$. After using n units of numerical ressources the algorithm $\mathcal{A}$ returns the action $\mathcal{A}(n)$. The regret resulting from choosing this action instead of the optimal one is:

$$R_{\mathcal{A}}(n) \overset{\text{def}}{=} \max_{a \in A} Q^*(x,a) - Q^*(x, \mathcal{A}(n)). \tag{1}$$

From an action-selection algorithm $\mathcal{A}$ one may define a policy $\pi_{\mathcal{A}}$ which would select in each state encountered along a trajectory the action $\mathcal{A}(n)$ proposed by the algorithm $\mathcal{A}$ using n resources. The following result motivates our choice of the previous definition for the regret in the sense that an algorithm with small regret will generate a close-to-optimal policy.

Proposition 1. *Consider a control algorithm using an action-selection procedure with regret ϵ (i.e. for each state x, the action-selection procedure returns an action a such that $Q^*(x,a) \geq V^*(x) - \epsilon$). Then the performance of the resulting policy $\pi_{\mathcal{A}}$ is $\frac{\epsilon}{1-\gamma}$-optimal, i.e. for all x,*

$$V^*(x) - V^{\pi_{\mathcal{A}}}(x) \leq \frac{\epsilon}{1-\gamma}$$

Proof. Let T and T^π be operators over bounded functions on X, defined as follows: for any bounded $V : X \to \mathbb{R}$, $TV(x) \overset{\text{def}}{=} \max_{a \in A} \big[r(x,a) + \gamma V(f(x,a)) \big]$, $T^\pi V(x) \overset{\text{def}}{=} r(x, \pi(x)) + \gamma V(f(x, \pi(x)))$.

We have the properties that T and T^π are contraction operators in sup norm, and that V^* is the fixed point of T (i.e. $V^* = TV^*$) and V^π is the fixed point of T^π (i.e. $V^\pi = T^\pi V^\pi$), see e.g. [Put94].

Using these notations, the assumption that $\mathcal{A}$ is an action selection procedure with regret ϵ writes that for all x, $T^{\pi_{\mathcal{A}}} V^*(x) \geq TV^*(x) - \epsilon$. Thus we have: $||V^* - V^{\pi_{\mathcal{A}}}||_\infty \leq ||TV^* - T^{\pi_{\mathcal{A}}} V^*||_\infty + ||T^{\pi_{\mathcal{A}}} V^* - T^{\pi_{\mathcal{A}}} V^{\pi_{\mathcal{A}}}||_\infty \leq \epsilon + \gamma ||V^* - V^{\pi_{\mathcal{A}}}||_\infty$, from which we deduce the proposition. □

Our goal is thus to define clever action-selection algorithms $\mathcal{A}$ such that, if provided with n units of computational resource, would return an action with minimal regret $R_{\mathcal{A}}(n)$.

The next section describes a method based on the construction of a uniform look-ahead tree.

3 Uniform Planning

3.1 Look-Ahead Tree Search

Given a state x, we describe a way to select an almost-optimal action, based on the construction of a uniform look-ahead tree. We consider the (infinite) tree $\mathcal{T}$ composed of all possible reachable states from x: the root corresponds to x and each node of depth d correspond to a state that is reachable from x after a sequence of d transitions. Each node i (associated to some state y), has K children $\mathcal{C}(i)$ (associated with the states $\{f(y,a)\}_{a\in A}$). Write 0 the root node, and $1\dots K$ its K children (nodes of depth 1).

We call a path of $\mathcal{T}$ a (finite or infinite) sequence of connected nodes starting from the root. We define the value v_i of a node i as the supremum, over all infinite paths going through node i, of the sum of discounted rewards obtained along such paths. We have the property that $v_i = \max_{j\in\mathcal{C}(i)} v_j$, and the optimal value (value of the root) $v^* = v_0 = \max_{i\in\mathcal{T}} v_i = \max\{v_1,\dots,v_K\}$.

We consider numerical resources expressed in terms of the number of expanded nodes. This is directly related to the CPU time required to explore the corresponding part of the tree, or the amount of memory required to store information about the expanded nodes. This is also equivalent to the number of calls to the generative-model, providing the next-state $f(x,a)$ and the reward $r(x,a)$.

We say that a node is expanded when some numerical resources are allocated to this node and to the computation of the transitions to its children (by a call to the generative model for each actions). At any round n, the expanded tree $\mathcal{T}_n$ denotes the set of nodes already expanded. The set of nodes in $\mathcal{T}$ that are not in $\mathcal{T}_n$ but whose parents are in $\mathcal{T}_n$ is written $\mathcal{S}_n$: this represents the set of possible nodes that may be expanded at the next round (see fig. 1).

For any node $i\in\mathcal{S}_n$, we define the value u_i to be the sum of discounted rewards obtained along the (finite) path from the root to node i (this information

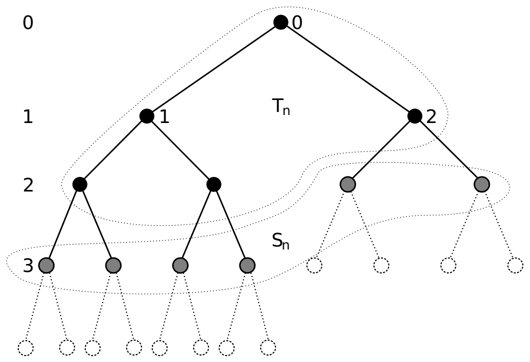


Fig. 1. Set of expanded nodes $\mathcal{T}_n$ (black dots) at round $n=5$ and set of nodes $\mathcal{S}_n$ (gray dots) that may be expanded next. Here $K=2$.

is available at round n since the parent of i has been expanded and thus the transition to node i has been computed). Now, for any node $i \in \mathcal{T}_n$, we define in a recursive way $u_i = \max_{j \in \mathcal{C}(i)} u_j$. Since these u-values are defined in $\mathcal{S}_n$, they are also well defined in $\mathcal{T}_n$.

Note that since the u-values depend on $\mathcal{T}_n$, we will write them $u_i(n)$ whenever the dependency with respect to (w.r.t.) n is relevant. From their definition, we have the property that $u_i(n)$ is an increasing function of n.

Since the sum of discounted rewards from a node of depth d is at most $\gamma^d + \gamma^{d+1} + \cdots = \frac{\gamma^d}{1-\gamma}$ (recall that all rewards are in $[0, 1]$), we have: for all $i \in \mathcal{T}_t \cup \mathcal{S}_t$ and $n \geq t \geq 1$, $u_i(n) \leq v_i \leq u_i(n) + \frac{\gamma^d}{1-\gamma}$.

3.2 The Algorithm

Here we consider a uniform exploration policy, defined as follows. We first expand the root. Then, at each round n, we expand a node in $\mathcal{S}_n$ having the smallest depth. See Algorithm 1.

Algorithm 1. Uniform planning algorithm $\mathcal{A}_U$

```
Set $n = 0$. Expand the root.
while Numerical resource available do
  Expand a node $i \in \mathcal{S}_n$ with the smallest depth.
  $n = n + 1$
end while
return A ction $\arg\max_{k \in \{1,\ldots,K\}} u_k(n)$
```

Thus, at all rounds, a uniform tree is expanded: Hence, at round $n = 1 + K + K^2 + \cdots + K^d = \frac{K^{d+1}-1}{K-1}$, all nodes of depth d or less have been expanded.

The uniform action-selection algorithm, written $\mathcal{A}_U$, returns, after n rounds, the action which corresponds to the children of the root $k \in \{1 \ldots K\}$ that has the highest u_k, i.e.:

$$\mathcal{A}_U(n) \overset{\text{def}}{=} \arg\max_{k \in \{1,\ldots,K\}} u_k(n),$$

(ties broken arbitrarily). In words, after n rounds, where we expanded the tree uniformly, $\mathcal{A}_U$ selects the action corresponding to the branch where we found the (finite) path with highest sum of rewards.

3.3 Analysis

Theorem 1. *Consider the uniform planning algorithm described above. Then for any reward function, the regret of the uniform algorithm is bounded by*

$$R_{\mathcal{A}_U}(n) \leq \frac{1}{\gamma(1-\gamma)} \left[n(K-1) + 1 \right]^{-\frac{\log 1/\gamma}{\log K}}. \tag{2}$$

Moreover, for all $n \geq 2$, there exists a reward function, such that the regret of this algorithm on this problem is at least

$$R_{\mathcal{A}_U}(n) \geq \frac{\gamma}{1-\gamma}\big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}. \tag{3}$$

We deduce the dependency (in a worst-case sense): $R_{\mathcal{A}_U}(n) = \Theta(n^{-\frac{\log 1/\gamma}{\log K}})$.

Proof. For any $n \geq 2$, let d be the largest integer such that

$$n \geq \frac{K^{d+1}-1}{K-1}. \tag{4}$$

Thus all nodes of depth d have been expanded. Thus for all $k \in \{1, \dots, K\}$, we have $v_k \leq u_k + \frac{\gamma^{d+1}}{1-\gamma}$, since all rewards up to depth d have been seen. Thus:

$$v^* = \max_{k \in \{1,\dots,K\}} v_k \leq \max_{k \in \{1,\dots,K\}} u_k + \frac{\gamma^{d+1}}{1-\gamma} = u_{\mathcal{A}_U(n)} + \frac{\gamma^{d+1}}{1-\gamma} \leq v_{\mathcal{A}_U(n)} + \frac{\gamma^{d+1}}{1-\gamma}.$$

Now, from (4), we have $d \geq \log_K[n(K-1)+1] - 2$, from which we deduce that

$$v^* - v_{\mathcal{A}_U(n)} \leq \frac{\gamma^{d+1}}{1-\gamma} \leq \frac{1}{\gamma(1-\gamma)}\big[n(K-1)+1\big]^{-\log_K 1/\gamma}.$$

For the second part of the Theorem, consider a fixed n. Define d as the largest integer such that (4) holds. Thus, from (4), we have $d \leq \log_K[n(K-1)+1]-1$. Define the following reward function: all rewards are 0 except in a branch (say branch 1) where the rewards of all transitions from nodes of depth p to depth $p+1$, where $p > d+1$, is 1. Thus $v^* = v_1 = \frac{\gamma^{d+2}}{1-\gamma}$ and $v_2 = 0$.

Thus at the time of the decision, all u_k (for $k \in \{1, \dots, K\}$) equal zero (only 0 rewards have been observed), and an arbitrary action (say action 2) is selected by the algorithm.

Thus $v^* - v_{\mathcal{A}_U(n)} = v_1 - v_2 = \frac{\gamma^{d+2}}{1-\gamma}$, and (3) follows from the bound on d. □

As a consequence of Theorem 1 and Proposition 1, we deduce that in order to guarantee that the performance of the uniform planning policy $\pi_{\mathcal{A}(n)}$ is ϵ-optimal (i.e. $||V^* - V^{\pi_{\mathcal{A}(n)}}||_\infty \leq \epsilon$), we need to devote $n = \Theta\big(\frac{1}{\epsilon(1-\gamma)^2}\big)^{\frac{\log K}{\log 1/\gamma}}$ units of resource per decision-step.

In this paper, we sought for other action-selection algorithms $\mathcal{A}$ that could make better use of available resources in the sense of minimizing the regret $R_{\mathcal{A}}(n)$ for a fixed amount n of computational resources. Note that the worst-case analysis considered in the second part of the proof of the previous Theorem may discourage us for searching for better bounds, since a similar analysis could be pursued on any specific algorithm. However we would like to define relevant classes of problems for which one would expect to obtain better convergence rates (for the regret) when using other action-selection algorithms than uniform: One cannot hope to achieve better rates for all problems but this may be possible for specific classes of problems.

4 Optimistic Planning

In this section, we present an algorithm building an asymmetric look-ahead tree aiming at exploring first the most promising parts of the tree.

4.1 The Algorithm

We define the b-value of each node $i \in \mathcal{S}_n$ of depth d by $b_i \overset{\text{def}}{=} u_i + \frac{\gamma^d}{1-\gamma}$, and for any node $i \in \mathcal{T}_n$, its b-value is defined recursively by: $b_i \overset{\text{def}}{=} \max_{j \in \mathcal{C}(i)} b_j$.

Note that like the u-values, the b-values are also well-defined and also depend on n, so we will write them $b_i(n)$ whenever the explicit dependency w.r.t. n is relevant. A property is that $b_i(n)$ is a decreasing function of n. Now, since all rewards are assumed to be in $[0, 1]$, we have the immediate property that these b-values are upper-bounds on the values of the nodes: for all $i \in \mathcal{T}_t \cup \mathcal{S}_t$, for all $n \geq t$,

$$u_i(n) \leq v_i \leq b_i(n).$$

The optimistic exploration policy consists in expanding in each round a node $i \in \mathcal{S}_n$ which possesses the highest b-value. See Algorithm 2. The returned action corresponds to the child of the root with highest u-value: $\mathcal{A}_O(n) \overset{\text{def}}{=} \arg\max_{k \in \{1,\dots,K\}} u_k(n)$ (ties broken arbitrarily).

Algorithm 2. Optimistic planning algorithm $\mathcal{A}_O$

```
Set $n = 0$. Expand the root.
while Numerical resource available do
    Expand a node $i \in \mathcal{S}_n$ s.t. $\forall j \in \mathcal{S}_n$, $b_i(n) \geq b_j(n)$.
    $n = n + 1$
end while
return A ction $\arg \max_{k \in \{1,\dots,K\}} u_k(n)$
```

4.2 Analysis

Theorem 2. *Consider the optimistic planning algorithm described above. At round n, the regret is bounded by*

$$R_{\mathcal{A}_O}(n) \leq \frac{\gamma^{d_n}}{1-\gamma}, \tag{5}$$

where d_n is the depth of the expanded tree $\mathcal{T}_n$ (maximal depth of nodes in $\mathcal{T}_n$).

As a consequence, for any reward function, the upper bound on the regret for the optimistic planning is never larger than that of the uniform planning (since the uniform exploration is the exploration strategy which minimizes the depth d_n for a given n, the depth obtained when using an optimistic algorithm is at least as high as that of the uniform one).

Proof. First let us notice that the action returned by the optimistic algorithm corresponds to a deepest explored branch. Indeed, if this was not the case, this would mean that if we write $i \in \mathcal{T}_n$ a node of maximal depth d_n, there exists a node $j \in \mathcal{T}_n$ of depth $d < d_n$ such that $u_j(n) \geq u_i(n)$. Thus there would exist a round $t \leq n$ such that node i has been expanded at round t, which would mean that $b_i(t) \geq b_j(t)$. But this is impossible since $b_i(t) = u_i(t) + \frac{\gamma^{d_n}}{1-\gamma} \leq u_i(n) + \frac{\gamma^{d_n}}{1-\gamma} \leq u_j(n) + \frac{\gamma^{d_n}}{1-\gamma} < u_j(n) + \frac{\gamma^{d}}{1-\gamma} = b_j(n) \leq b_j(t)$.

Thus the action returned by the algorithm corresponds to a branch that has been the most deeply explored. Let $i \in \mathcal{T}_n$ be a node of maximal depth d_n. Node i belongs to one of the K branches connected to the root, say branch 1. If the optimal action is 1 then the loss is 0 and the upper bound holds. Otherwise, the optimal action is not 1, say it is 2. Let $t \leq n$ be the round at which the node i was expanded. This means that $b_i(t) \geq b_j(t)$ for all nodes $j \in \mathcal{S}_t$, thus also for all nodes $j \in \mathcal{T}_t$. In particular, $b_1(t) = b_i(t) \geq b_2(t)$. But $b_2(t) \geq b_2(n)$. Now, the u-values are always lower bounds on the v-values, thus $u_1(t) \leq v_1$, and from the definition of u-values, $u_i(t) \leq u_1(t)$. We thus have:

$$\begin{aligned} v^* - v_{\mathcal{A}_O(n)} &= v_2 - v_1 \leq b_2(n) - v_1 \leq b_2(t) - v_1 \\ &\leq b_1(t) - u_1(t) \leq b_i(t) - u_i(t) = \frac{\gamma^{d_n}}{1-\gamma}, \end{aligned}$$

which concludes the proof. □

Remark 1. *This result shows that the upper bound for optimistic planning cannot be worst than the upper bound for uniform planning. This does not mean that for any problem, the optimistic algorithm will perform at least as well as a uniform algorithm. Indeed, this is not true since, if we consider the example mentioned in the proof of Theorem 1, if at time* n *all observed rewards are* 0*, any algorithm (such as the optimistic one) would deliver an arbitrary decision, which may be worst than another arbitrary decision (made for example by the uniform algorithm). However, if we consider equivalent classes of problems, where classes are defined by trees having the same reward function up to possible permutations of branches, then we conjecture that we have the stronger result that for any problem, the optimistic planning is never worse than the uniform planning performed on a problem of the same class. However we will not pursue this research further in this paper.*

Note that the lower bound obtained for the uniform planning also holds for the optimistic planning (the proof is the same: since, up to round n all rewards are 0, the optimistic planning will also build a uniform tree). This shows that no improvement (over uniform planning) may be expected in a worst-case setting, as already mentioned. In order to quantify possible improvement over uniform planning, one thus needs to define specific classes of problems.

For any $\epsilon \in [0, 1]$, define the proportion $p_d(\epsilon)$ of ϵ-optimal nodes of depth d:

$$p_d(\epsilon) \overset{\text{def}}{=} |\{\text{node } i \text{ of depth } d \text{ s.t. } v_i \geq v^* - \epsilon\}| K^{-d},$$

and write $p(\epsilon) \overset{\text{def}}{=} \lim_{d\to\infty} p_d(\epsilon)$ the proportion of ϵ-optimal paths in the tree (note that this limit is well defined since $p_d(\epsilon)$ is a decreasing function of d).

Theorem 3. *Let $\beta \in [0, \frac{\log K}{\log 1/\gamma}]$ be such that the proportion of ϵ-optimal nodes of depth $d \geq d_0$ (for some depth d_0) in the tree $\mathcal{T}$ is $O(\epsilon^\beta)$. More precisely, we assume that there exists positive constants d_0 and c, such that $\forall d \geq d_0$ $\forall \epsilon \geq 0$, $p_d(\epsilon) \leq c\epsilon^\beta$. Now let us define $\kappa \overset{\text{def}}{=} K\gamma^\beta$, which belongs to the interval $[1, K]$.*

If $\beta < \frac{\log K}{\log 1/\gamma}$ (i.e. $\kappa > 1$), then the regret of the optimistic algorithm is

$$R_{\mathcal{A}_O}(n) = O\big(n^{-\frac{\log 1/\gamma}{\log \kappa}}\big).$$

If $\beta = \frac{\log K}{\log 1/\gamma}$ (i.e. $\kappa = 1$), then we have the exponential rate:

$$R_{\mathcal{A}_O}(n) = O\big(\gamma^{\frac{(1-\gamma)^\beta}{c} n}\big).$$

Proof. Let us define the sub-tree $\mathcal{T}_\infty \subset \mathcal{T}$ of all the nodes i of depth d that are $\frac{\gamma^d}{1-\gamma}$-optimal, i.e.

$$\mathcal{T}_\infty \overset{\text{def}}{=} \bigcup_{d\geq 0} \big\{\text{node } i \text{ of depth } d \text{ s.t. } v_i + \frac{\gamma^d}{1-\gamma} \geq v^*\big\}.$$

Let us prove that all nodes expanded by the optimistic algorithm are in $\mathcal{T}_\infty$. Indeed, let i be a node of depth d expanded at round n. Then $b_i(n) \geq b_j$ for all $j \in \mathcal{S}_n \cup \mathcal{T}_n$, thus $b_i(n) = b_0(n)$ (b-value of the root). But $b_0(n) \geq v_0 = v^*$, thus $v_i \geq u_i(n) = b_i(n) - \frac{\gamma^d}{1-\gamma} \geq v^* - \frac{\gamma^d}{1-\gamma}$, thus $i \in \mathcal{T}_\infty$.

Now, from the definition of β, there exists d_0 such that the proportion of ϵ-optimal nodes of depth $d > d_0$ is at most $c\epsilon^\beta$, where c is a constant. We thus have that the number n_d of nodes of depth d in $\mathcal{T}_\infty$ is bounded by $c\big(\frac{\gamma^d}{1-\gamma}\big)^\beta K^d$.

Now, write d_n the depth of the expanded tree $\mathcal{T}_n$ at round n. Let $n_0 = \frac{K^{d_0+1}-1}{K-1}$ the number of nodes in $\mathcal{T}$ of depth less than d_0. We have:

$$n \leq n_0 + \sum_{d=d_0+1}^{d_n} n_d \leq n_0 + c \sum_{d=d_0+1}^{d_n} \big(\frac{\gamma^d}{1-\gamma}\big)^\beta K^d = n_0 + c' \sum_{d=d_0+1}^{d_n} \kappa^d$$

with $c' = c/(1-\gamma)^\beta$ and $\kappa = \gamma^\beta K$.

First, if $\kappa > 1$ then we have $n \leq n_0 + c'\kappa^{d_0+1}\frac{\kappa^{d_n-d_0}-1}{\kappa-1}$. Thus $d_n \geq d_0 + \log_K \frac{(n-n_0)(\kappa-1)}{c'\kappa^{d_0+1}}$. Now, from Theorem 2, we have the regret:

$$\begin{aligned} R_{\mathcal{A}_O}(n) &\leq \frac{\gamma^{d_n}}{1-\gamma} = \frac{1}{1-\gamma}\left[\frac{(n-n_0)(\kappa-1)}{c'\kappa^{d_0+1}}\right]^{\frac{\log\gamma}{\log\kappa}} \\ &= O\big(n^{-\frac{\log 1/\gamma}{\log\kappa}}\big). \end{aligned}$$

Now, if $\kappa = 1$, let $n_0 = \frac{K^{d_0+1}-1}{K-1}$. Following the same arguments as above, we deduce that $n \leq n_0 + \frac{c}{(1-\gamma)^\beta}(d_n - d_0)$, and the regret: $R_{\mathcal{A}_O}(n) \leq \frac{\gamma^{d_n}}{1-\gamma} = O\big(\gamma^{n\frac{(1-\gamma)^\beta}{c}}\big)$. □

Remark 2. *Let us notice that the proportion of ϵ-optimal paths is bounded by $c\epsilon^\beta$. β lies necessarily in the interval $[0, \log_{1/\gamma} K]$, and two extreme cases are:*

- *The case when all paths are optimal (i.e. all rewards are equal). This corresponds to $\beta = 0$, or $\kappa = K$.*
- *The case where there is only one path where rewards are 1, all other rewards being 0. Then, for any ϵ, the proportion of ϵ-optimal nodes of depth d is $1/K^d$ for $d \le d_0$ for some depth d_0 for which $\frac{\gamma^{d_0}}{1-\gamma} \le \epsilon$, and for all $d > d_0$ the proportion of ϵ-optimal nodes remains constant. Since $d_0 \ge \log_\gamma (1-\gamma)\epsilon$, the proportion of ϵ-optimal nodes of depth $d > d_0$ is at most $\big[(1-\gamma)\epsilon\big]^{\log_{1/\gamma} K}$, i.e. which corresponds to $\beta = \log_{1/\gamma} K$. This is the highest possible value for β. This corresponds to $\kappa = 1$.*

From this result, we see that when $\kappa > 1$, the decrease rate of the regret for the optimistic algorithm is $n^{-\frac{\log 1/\gamma}{\log \kappa}}$ instead of the $n^{-\frac{\log 1/\gamma}{\log K}}$ for the uniform one. By looking at the proof, we observe that κ plays the role of the **branching factor of the tree** $\mathcal{T}_\infty$ of the expandable nodes, similarly to K being the branching factor of the full tree $\mathcal{T}$. κ belongs to the interval $[1, K]$, thus the decrease rate for the regret of the optimistic planning is always at least as good as that of the uniform one. The bound for the optimistic planning is better than uniform whenever $\kappa < K$, and greatly better when κ is close to 1. Note that the tree $\mathcal{T}_\infty$ represents the set of nodes i such that given the observed rewards from the root to node i, one cannot decide whether i lies in an optimal path or not. This represents the set of nodes which would have to be expanded for finding an optimal path. Thus the performance of the optimistic algorithm is expressed in terms of the branching factor of the subtree $\mathcal{T}_\infty$ composed of all the nodes that have to be expanded eventually. We believe this is a strong optimality result, although we do not have lower bounds expressed in terms of β, yet.

Remark 3. *The case $\kappa = 1$ is also interesting because it provides rates that are exponential in n instead of polynomial ones. This case would hold for example if there exists d_0 such that for each node i_d of depth $d \ge d_0$ along an optimal path, if we write x_d the state corresponding to that node i_d, we require that the gap between the optimal value function at x_d and the Q-value of all suboptimal actions are larger than some constant value $\Delta > 0$, i.e., $\exists \Delta > 0$, for all $d \ge d_0$,*

$$V^*(x_d) - \max_{a \text{ s.t. } Q^*(x_d,a) < V^*(x_d)} Q^*(x_d, a) \ge \Delta. \tag{6}$$

Indeed, if this was true, we would have for all $d \ge d_0$, for any non-optimal child j of i_d, $v_j \le v^* - \Delta\gamma^d$ (since $v_{i_d} = v^*$ because the nodes i_d are optimal). Now, the number of nodes in the branch j that belong to the expandable tree $\mathcal{T}_\infty$ is bounded by K^{h-d-1} where h is the maximal depth such that $v_j + \frac{\gamma^h}{1-\gamma} \ge v^*$, i.e., $\gamma^{h-d} \ge \Delta(1-\gamma)$. Thus $K^{h-d-1} = \frac{1}{K}\big[(1-\gamma)\Delta\big]^{-\beta}$ with $\beta = \log K / \log(1/\gamma)$.

Thus the number of nodes of depth $d \ge d_0$ in $\mathcal{T}_\infty$ is bounded by a constant independent of d, i.e. $\frac{1}{K}[\Delta(1-\gamma)]^{-\beta}$. Thus $p_d(\frac{\gamma^d}{1-\gamma}) \le \frac{1}{K}\big[\Delta(1-\gamma)\big]^{-\beta}/K^d$. Thus

$p_n(\epsilon) \leq \frac{1}{K}\left[\epsilon/\Delta\right]^\beta$. Thus $p(\epsilon) \leq \frac{1}{K}\left[\epsilon/\Delta\right]^\beta$ and we have $\kappa = 1$ and the constant $c = \frac{1}{K}(1/\Delta)^\beta$.

Remark 4. *A natural extension of the previous case (for which we deduced $\kappa = 1$) is when from each state x_d corresponding to a node of depth $d \geq d_0$, for some d_0, there exist a small number m of Δ-optimal sequences of h actions, where $\Delta > 0$ is a fixed constant and h a fixed integer. This means that from x_d there exist at most m sequences of actions $a_1, \ldots, a_h$, leading to states $(x_{d+i})_{1\leq i\leq h}$, such that*

$$\sum_{i=0}^{h-1} \gamma^i r(x_{d+i}, a_{i+1}) + \gamma^h V^*(x_{d+h}) \geq V^*(x_d) - \Delta.$$

Then one can prove that the branching factor κ of the expandable tree is at most $m^{1/h}$. Notice that in the case presented in the previous remark we had $m = 1$ and $h = 1$ (and we deduced that $\kappa = 1$). However note that in the previous remark, we only assumed the property (6) along the optimal path (whereas we impose here that it holds for all nodes). The proof of this result is rather technical and not included here.

5 Numerical Experiments

We have done some numerical experiments to compare uniform and optimistic planning algorithms. We consider the system $\ddot{y}_t = a_t$, where a point defined by its position y_t and its velocity v_t is controlled by a force (action) $a_t \in \{-1, 1\}$. The dynamics are: $(y_{t+1}, v_{t+1})' = (y_t, v_t)' + (v_t, a_t)'\Delta t$, where $\Delta t = 0.1$ is the time discretization step. The reward of state (y_t, v_t) action a_t is defined by $\max(1 - y_{t+1}^2, 0)$ where y_{t+1} is the position of the resulting next state.

In figure 2 we show the trees resulting from the uniform and optimistic algorithms using the same number of numerical resources $n = 3000$. The initial state is $(y_0 = -1, v_0 = 0)$ and the discount factor $\gamma = 0.9$. The optimistic tree is not deeply explored on the left side of the initial state since this corresponds to states with low rewards, however it is more deeply expanded along some branches which provide high rewards, and in particular, the branch leading the states around the origin is very deeply (and sharply) expanded (maximal depth of 49).

We computed an average regret of both algorithms for $n = 2^{d+1} - 1$ with $d \in \{2, \ldots, 18\}$ (full trees of depth d), where the average is performed over 1000 trees where the initial state have been uniformly randomly sampled in the domain $[-1, 1] \times [-2, 2]$. Figure 3 shows the regret (in log scales) for both algorithms. The slope $\frac{\log R(n)}{\log n}$ of these curves indicate the exponent in the regret polynomial dependency. For the uniform curve, we calculate a numerical slope of about -1, thus the regret $R_{\mathcal{A}_U}(n) \simeq 1/n$. For the optimistic curve, the numerical slope is about $-3 = -\frac{\log 1/\gamma}{\kappa}$ which corresponds to the branching factor $\kappa = 1.04$. The regret of optimistic planning is thus of order $1/n^3$ which is significantly a better rate than the uniform one, and explains the great improvement of the optimistic

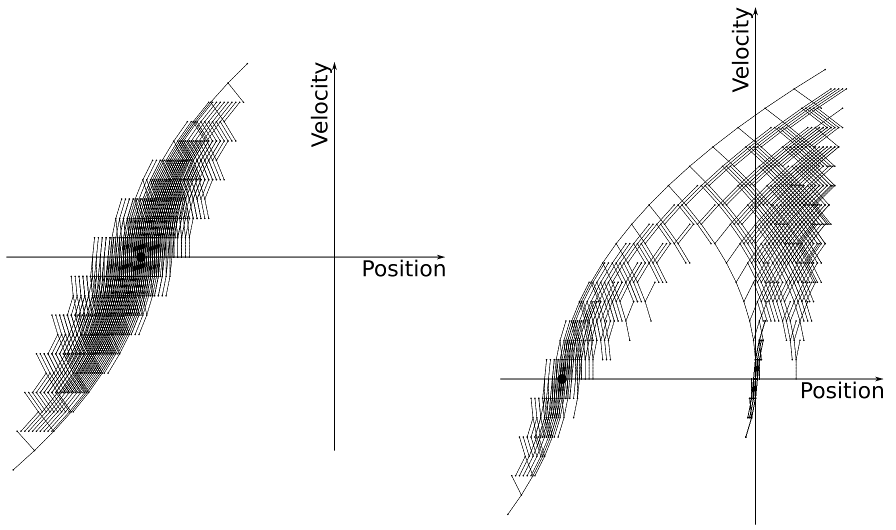


Fig. 2. Expanded trees with $n = 3000$ calls to the generative model using the uniform (left figure) and optimistic (right) planning algorithms. The depth of the trees is 11 for the uniform and 49 for the optimistic.

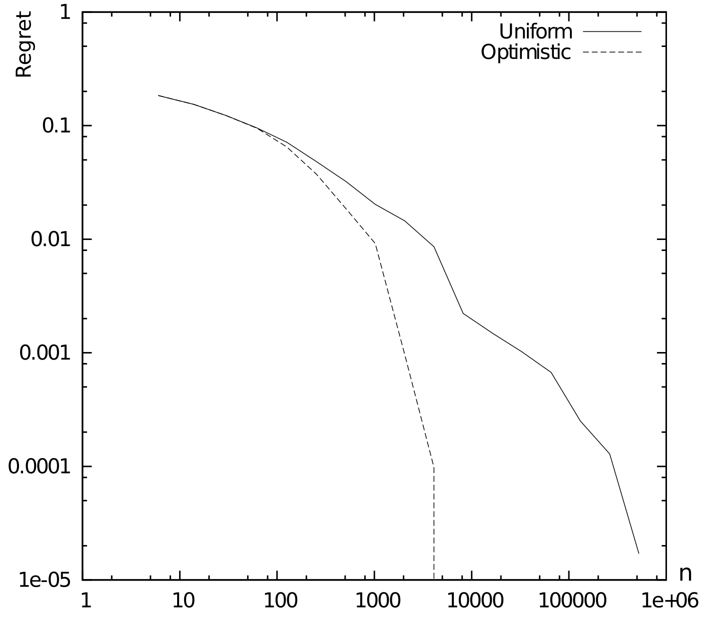


Fig. 3. Average regret $R(n)$ for both algorithms as a function of $n = 2^{d+1} - 1$ with $d \in \{2, \ldots, 18\}$, in logarithmic scales. The slope of each curve indicates the exponent in the actual polynomial dependency of the regret.

planning over the uniform approach. For illustration, achieving a regret of 0.0001 with uniform planning requires more than $n = 262142$ expanded nodes whereas optimistic planning only requires $n = 4094$ such nodes.

We do not have space to describe more challenging applications here but other simulations, including the double inverted pendulum linked by a spring, are described at the address: `http://sequel.futurs.inria.fr/hren/optimistic/`.

6 Conclusions and Future Works

An immediate remaining work is the derivation of β-dependant lower bounds for the optimistic planning. An extension of this work would consider the case of stochastic rewards, like in [CM07], where an additional term (coming from a Chernoff-Hoeffing bound) would be added to the b-values to define high probability upper-confidence bounds. Another, more challenging, extension would consider the stochastic transitions case. The possibility of combining this pure search approach with local approximation of the optimal value function is certainly worth investigating too.

References

[ACBF02] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning Journal 47(2-3), 235–256 (2002)

[BT96] Bertsekas, D.P., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)

[CM07] Coquelin, P.-A., Munos, R.: Bandit algorithms for tree search. In: Uncertainty in Artificial Intelligence (2007)

[GWMT06] Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo go. Technical Report INRIA RR-6062 (2006)

[KMN02] Kearns, M., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markovian decision processes. Machine Learning 49, 193–208 (2002)

[KS06] Kocsis, L., Szepesvari, C.: Bandit based monte-carlo planning. In: European Conference on Machine Learning, pp. 282–293 (2006)

[LR85] Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics 6, 4–22 (1985)

[PG04] Péret, L., Garcia, F.: On-line search for solving large Markov decision processes. In: Proceedings of the 16th European Conference on Artificial Intelligence (2004)

[Put94] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, Chichester (1994)

[Rob52] Robbins, H.: Some aspects of the sequential design of experiments. Bulletin of the American Mathematics Society 58, 527–535 (1952)

[SB98] Sutton, R., Barto, A.: Reinforcement Learning. MIT Press, Cambridge (1998)

Policy Iteration for Learning an Exercise Policy for American Options

Yuxi Li and Dale Schuurmans

Department of Computing Science
University of Alberta
{yuxi,dale}@cs.ualberta.ca

Abstract. Options are important financial instruments, whose prices are usually determined by computational methods. Computational finance is a compelling application area for reinforcement learning research, where hard sequential decision making problems abound and have great practical significance. In this paper, we investigate reinforcement learning methods, in particular, least squares policy iteration (LSPI), for the problem of learning an exercise policy for American options. We also investigate a method by Tsitsiklis and Van Roy, referred to as FQI. We compare LSPI and FQI with LSM, the standard least squares Monte Carlo method from the finance community. We evaluate their performance on both real and synthetic data. The results show that the exercise policies discovered by LSPI and FQI gain larger payoffs than those discovered by LSM, on both real and synthetic data. Our work shows that solution methods developed in reinforcement learning can advance the state of the art in an important and challenging application area, and demonstrates furthermore that computational finance remains an under-explored area for deployment of reinforcement learning methods.

1 Introduction

Options are an essential financial instrument for hedging and risk management, and therefore, options pricing and finding optimal exercise policies are important problems in finance.[1] Options pricing is usually approached by computational methods. In general, computational finance is a compelling application area for reinforcement learning research, where hard sequential decision making problems abound and have great practical significance [11]. In this paper, we show solution techniques from the reinforcement learning literature are superior to a standard technique from the finance literature for pricing American options, a classical sequential decision making problem in finance.

Options pricing is an optimal control problem, usually modeled as Markov Decision Processes (MDP). Dynamic programming is a method of finding an

[1] A call/put option gives the holder the right, not the obligation, to buy/sell the underlying asset, for example, a share of a stock, by a certain date (maturity date) for a certain price (strike price). An American option can be exercised any time up to the maturity date.

optimal policy for an MDP [2, 12], usually with the model of the MDP. Reinforcement learning, also known as neuro-dynamic programming, can work without a model of the MDP [3, 13]. When the size of an MDP is large, for example, when the state space is continuous, we encounter the "curse of dimensionality". Successful investigations include the application of reinforcement learning to playing backgammon, dynamic channel allocation, elevator dispatching, and so on. The key idea behind these successes is to exploit effective approximation methods. Linear approximation has been most widely used. A reinforcement learning method can learn an optimal policy for an MDP either from simulated samples or directly from real data. One advantage of basing directly an approximation architecture on the underlying MDP is that the error for the simulation model is eliminated.

In the community of computational finance, researchers have investigated pricing methods using analytic models and numerical methods, including the risk-neutral approach, the lattice and finite difference methods, and the Monte Carlo methods. For example, Hull [8] provides an introduction to options and other financial derivatives and their pricing methods, Broadie and Detemple [5] survey option pricing methods, and Glasserman [7] provides a book length treatment for Monte Carlo methods. Most of these methods follow the backward-recursive approach of dynamic programming. Two examples that deploy approximate dynamic programming for the problem of pricing American options are: the least squares Monte Carlo (LSM) method in [10] and the approximate value iteration approach in [14].

Our goal is to investigate reinforcement learning type algorithms for pricing American options. In this work, we extend an approximate policy iteration method, namely, least squares policy iteration (LSPI) in [9], to the problem of pricing American options. We also investigate the method proposed in [14], referred to as FQI. We empirically evaluate the performance of LSPI, FQI and LSM, with respect to the payoffs the exercise policies gain. In contrast, previous work evaluates pricing methods by measuring the accuracy of the estimated prices. The results show that, on both real and synthetic data, exercise policies discovered by LSPI and FQI can achieve larger payoffs than those found by LSM.

In this work, we present a successful application of reinforcement learning research, the policy iteration method, for learning an exercise policy for American options, and show its superiority to LSM, the standard option pricing method in finance. As well, we introduce a new performance measure, the payoff a pricing method gains, for comparing option pricing methods in the empirical study.

The remainder of this paper is organized as follows. First, we introduce MDPs and LSPI. Then, we present the extension of LSPI to pricing American options, and introduce FQI and LSM. After that, we study empirically the performance of LSPI, FQI and LSM on both real and synthetic data. Finally, we conclude.

2 Markov Decision Processes

The problem of sequential decision making is common in economics, science and engineering. Many of these problems can be modeled as MDPs. An MDP is defined by the 5-tuple (S, A, P, R, γ). S is a set of states; A is a set of actions; P is a transition model, with $P(s, a, s')$ specifying the conditional probability of transitioning to state s' starting from state s and taking action a; R is a reward function, with $R(s, a, s')$ being the reward for transition to state s' starting from state s and taking action a; and γ is a discount factor.

A policy π is a rule for selecting actions based on observed states. $\pi(s, a)$ specifies the probability of selecting action a in state s by following policy π. An optimal policy maximizes the rewards obtained over the long run. We define the long run reward in an MDP as maximizing the infinite horizon discounted reward $\sum_{t=0}^{\infty} \gamma^t r_t$ obtained over an infinite run of the MDP, given a discount factor $0 < \gamma < 1$. A policy π is associated with a value function for each state-action pair (s, a), $Q^\pi(s, a)$, which represents the expected, discounted, total reward starting from state s taking action a and following policy π thereafter. That is, $Q^\pi(s, a) = E(\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a)$, where the expectation is taken with respect to policy π and the transition model P. Q^π can be found by solving the following linear system of Bellman equations: $Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \sum_{a' \in A} \pi(s', a') Q^\pi(s', a')$, where $R(s, a) = \sum_{s'} P(s, a, s') R(s, a, s')$ is the expected reward for state-action pair (s, a). Q^π is the fixed point of the Bellman operator T_π: $(T_\pi Q)(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \sum_{a' \in A} \pi(s', a') Q(s', a')$. T_π is a monotonic operator and a contraction mapping in the L_∞-norm. The implication is that successive application of T_π for any initial Q converges to Q^π. This is value iteration, a principled method for computing Q^π.

When the size of an MDP becomes large, its solution methods encounter the "curse of dimensionality". Approximation architecture is an approach to addressing the scalability concern. The linear architecture is an efficient and effective approach. In the linear architecture, the approximate value function is represented by: $\hat{Q}^\pi(s, a; w) = \sum_{i=1}^{k} \phi_i(s, a) w_i$, where $\phi_i(\cdot, \cdot)$ is a basis function, w_i is its weight, and k is the number of basis functions.[2] Define

$$\phi(s,a) = \begin{pmatrix} \phi_1(s,a) \\ \phi_2(s,a) \\ \cdots \\ \phi_k(s,a) \end{pmatrix}, \mathbf{\Phi} = \begin{pmatrix} \phi(s_1,a_1)^\mathsf{T} \\ \cdots \\ \phi(s,a)^\mathsf{T} \\ \cdots \\ \phi(s_{|S|},a_{|A|})^\mathsf{T} \end{pmatrix}, w^\pi = \begin{pmatrix} w_1^\pi \\ w_2^\pi \\ \cdots \\ w_k^\pi \end{pmatrix},$$

where T denotes matrix transpose. $\hat{Q}^\pi$ then can be represented as $\hat{Q}^\pi = \mathbf{\Phi} w^\pi$.

Least squares policy iteration. Policy iteration is a method of discovering an optimal solution for an MDP. LSPI [9] combines the data efficiency of the

[2] Following the conventional notation, an approximate representation is denoted with the ˆ symbol, and a learned estimate is denoted with the ˜ symbol.

least squares temporal difference method [4] and the policy search efficiency of policy iteration. Next, we give a brief introduction of LSPI.[3] The matrix form of the Bellman equation is: $Q^\pi = R + \gamma \mathbf{P}\mathbf{\Pi}_\pi Q^\pi$, where $\mathbf{P}$ is a $|S||A| \times |A|$ matrix, with $\mathbf{P}((s,a),s') = P(s,a,s')$, and $\mathbf{\Pi}$ is a $|S| \times |S||A|$ matrix, with $\mathbf{\Pi}(s',(s',a')) = \pi(s',a')$.

The state-action value function Q^π is the fixed point of the Bellman operator: $T_\pi Q^\pi = Q^\pi$. An approach to finding a good approximation is to force $\hat{Q}^\pi$ to be a fixed point of the Bellman operator: $T_\pi \hat{Q}^\pi \approx \hat{Q}^\pi$. $\hat{Q}^\pi$ is in the space spanned by the basis functions. However, $T_\pi \hat{Q}^\pi$ may not be in this space. LSPI requires that, $\hat{Q}^\pi = \mathbf{\Phi}(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathsf{T}(T_\pi \hat{Q}^\pi) = \mathbf{\Phi}(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathsf{T}(R + \gamma \mathbf{P}\mathbf{\Pi}_\pi Q^\pi)$, where, $\mathbf{\Phi}(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathsf{T}$ is the orthogonal projection which minimizes the L_2-norm. From this, we can obtain, $w^\pi = \left(\mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi})\right)^{-1} \mathbf{\Phi}^\mathsf{T} R$. The weighted least squares fixed point solution is: $w^\pi = \left(\mathbf{\Phi}^\mathsf{T}\Delta_\mu(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi})\right)^{-1} \mathbf{\Phi}^\mathsf{T}\Delta_\mu R$, where Δ_μ is the diagonal matrix with the entries of $\mu(s,a)$, which is a probability distribution over state-action pairs ($S \times A$). This can be written as $\mathbf{A}w^\pi = b$, where $\mathbf{A} = \mathbf{\Phi}^\mathsf{T}\Delta_\mu(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi})$ and $b = \mathbf{\Phi}^\mathsf{T}\Delta_\mu R$.

Without a model of the MDP, that is, without the full knowledge of $\mathbf{P}$, $\mathbf{\Pi}_\pi$ and R, we need a learning method to discover an optimal policy. It is shown in [9] that $\mathbf{A}$ and b can be learned incrementally as, at iteration $t+1$:

$$\begin{aligned}\tilde{\mathbf{A}}^{(t+1)} &= \tilde{\mathbf{A}}^{(t)} + \phi(s_t,a_t)(\phi(s_t,a_t) - \gamma\phi(s'_t,a'_t))^\mathsf{T} \\ \tilde{b}^{(t+1)} &= \tilde{b}^{(t)} + \phi(s_t,a_t)R_t\end{aligned}$$

3 Learning an Exercise Policy for American Options

We first discuss the application of LSPI for the problem of learning an exercise policy for American options. Next we give a brief review of FQI [14] and LSM [10]. We discretize the time, thus the options become Bermudan. When each time step is very small, the results are close to those for American options.

3.1 LSPI for Learning an Exercise Policy for American Options

We need to consider several peculiarities of the problem of learning an exercise policy for American options, when applying LSPI for it. First, it is an episodic, optimal stopping problem. It may terminate any time between the starting date and the maturity date of the option. Usually, after a termination decision is made, LSPI needs to start over from a new sample path. This is data inefficient. We use the whole sample path, even in the case the option is exercised at an intermediate time step following the current policy. Second, in option pricing, the continuation value of an option may be different at different times, even with the same underlying asset price and other factors. Thus we incorporate time as a component in the state space. Third, there are two actions for each

[3] This is the LPSI with least-squares fixed-point approximation. LSPI can also work with Bellman residual minimizing approximation, which we do not discuss here.

state, exercise and continue. The state-action value function of exercising the option, that is, the intrinsic value of the option, can be calculated exactly. We only need to consider the state-action value function for continuation, that is, $Q(s, a = \text{continue})$. Fourth, before exercising an option, there is no reward to the option holder, that is, $R = 0$. When the option is exercised, the reward is the payoff.

3.2 FQI: The Policy Iteration Approach in [14]

We introduce FQI [14] in the following. We use $Q(S,t)$ to denote $Q(\{S,t\}, a = \textit{continue})$, where S is the stock price. We want to find a projection $\mathit{\Pi}$ of $Q = (Q(S,0), Q(S,1), \ldots, Q(S,T-1))$ in the form $\mathbf{\Phi}w$, where w is to minimize $\sum_{t=0}^{T-1} E[(\mathbf{\Phi}(S_t,t)w - Q(S_t,t))^2]$, where the expectation $E[(\mathbf{\Phi}(S_t,t)w - Q(S_t,t))^2]$ is with respect to the probability measure of S_t. The weight w is given by

$$w = \left(\sum_{t=0}^{T-1} E[\mathbf{\Phi}(S_t,t)\mathbf{\Phi}^{\mathsf{T}}(S_t,t)]\right)^{-1} E[\mathbf{\Phi}(S_t,t)Q(S_t,t)] \tag{1}$$

Define $g(S)$ as the intrinsic value of the option when the stock price is S, and $J_t(S)$ as the price of the option at time t when $S_t = S$: $J_T = g$ and $J_t = \max(g, \gamma\mathcal{P}J_{t+1}), t = T-1, T-2, \ldots, 0$, where $(\mathcal{P}J)(S) = E[J(S_{t+1})|S_t = S]$. Define $FJ = \gamma\mathcal{P}\max(g, J)$. We have $(Q(\cdot,0), Q(\cdot,1), \ldots, Q(\cdot,T-1)) = (FQ(\cdot,1), FQ(\cdot,2), \ldots, FQ(\cdot,T))$, which is denoted compactly as $Q = HQ$. The above solution of w would thus be the fixed point of the equation $HQ^* = Q^*$. However, it is difficult to solve this function, since Q^* is unknown. We resort to the fixed point of equation $Q = \mathit{\Pi}HQ$. Suppose w_i is the weight vector computed at iteration i (w_0 can be arbitrarily initialized),

$$w_{i+1} = \left(\sum_{t=0}^{T-1} E[\mathbf{\Phi}(S_t,t)\mathbf{\Phi}^{\mathsf{T}}(S_t,t)]\right)^{-1} E[\mathbf{\Phi}(S_t,t)\max(g(S_{t+1}), \mathbf{\Phi}(S^j_{t+1},t)w_i)] \tag{2}$$

The expectation with respect to the underlying probability measure can be replaced with an expectation with respect to the empirical measure provided by unbiased samples. The following is an implementable version with sample trajectories $S^j_t, j = 1, \ldots, m$, where S^j_t is the value of S_t in the j-th trajectory:

$$\hat{w}_{i+1} = \left(\sum_{t=0}^{T-1}\sum_{j=1}^{m} \mathbf{\Phi}(S^j_t,t)\mathbf{\Phi}^{\mathsf{T}}(S^j_t,t)\right)^{-1} \sum_{t=0}^{T-1}\sum_{j=1}^{m} \mathbf{\Phi}(S^j_t,t)\max(g(S^j_{t+1}), \mathbf{\Phi}(S^j_{t+1},t)\hat{w}_i) \tag{3}$$

3.3 Least Squares Monte Carlo

LSM in [10] follows the backward-recursive dynamic programming approach with function approximation of expected continuation value. It estimates the expected continuation value from the second-to-last time step backward until the first time step, on the sample paths. At each time step, LSM fits the expected continuation value on the set of basis functions with least squares regression, using the cross-sectional information from the sample paths and the previous iterations (or the last time step). Specifically, at time step t, assuming the option is not exercised, the continuation values for the sample paths (LSM uses only in-the-money paths) can be computed, since in a backward-recursive approach, LSM has already considered time steps after t until the maturity. As well, values of the basis functions can be evaluated for the asset prices at time step t. Then, LSM regresses the continuation values on the values of the basis functions with least squares, to obtain the weights for the basis functions for time step t. When LSM reaches the first time step, it obtains the price of the option. LSM also obtains the weights for the basis functions for each time step. These weights represent implicitly the exercising policy. The approximate value iteration method in [14] is conceptually similar to LSM. (FQI is also proposed in [14].)

4 Empirical Study

We study empirically the performance of LSPI, FQI and LSM on learning an exercise policy for American options. We study the plain vanilla American put stock options and American Asian options. We focus on at-the-money options, that is, the strike price is equal to the initial stock price. For simplicity, we assume the risk-free interest rate r is constant and stocks are non-dividend-paying. We assume 252 trading days in each year. We study options with quarterly, semi-annual and annual maturity terms, with 63, 126 and 252 days duration respectively. Each time step is one trading day, that is, 1/252 trading year. In LSPI, we set the discount factor $\gamma = e^{-r/252}$, corresponding to the daily interest rate. LSPI and FQI iterate on the sample paths until the difference between two successive policies is sufficiently small, or when it has run 15 iterations (LSPI and FQI usually converge in 4 or 5 iterations). We obtain five years' daily stock prices from January 2002 to December 2006 for Dow Jones 30 companies from WRDS, Wharton Research Data Services. We study the payoff a policy gains, which is the intrinsic value of an option when the option is exercised.

4.1 Simulation Models

In our experiments when a simulation model is used, synthetic data may be generated from either the geometric Brownian Motion (GBM) model or a stochastic volatility (SV) model, two of the most widely used models for stock price movement. See [8] for detail.

Geometric Brownian motion model. Suppose S_t, the stock price at time t, follows a GBM:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{4}$$

where, μ is the risk-neutral expected stock return, σ is the stock volatility and W is a standard Brownian motion. For a non-dividend-paying stock, $\mu = r$, the risk-free interest rate. It is usually more accurate to simulate $\ln S_t$ in practice. Using Itô's lemma, the process followed by $\ln S_t$ is:

$$d\ln\mathrm{S}_t = (\mu - \sigma^2/2)dt + \sigma dW_t. \tag{5}$$

We can obtain the following discretized version for (5), and use it to generate stock price sample paths:

$$S_{t+1} = S_t\exp\{(\mu - \sigma^2/2)\Delta t + \sigma\sqrt{\Delta t}\epsilon\}, \tag{6}$$

where Δt is a small time step, and $\epsilon \sim N(0,1)$, the standard normal distribution.

To estimate the constant σ from real data, we use the method of maximum likelihood estimation (MLE).

Stochastic volatility model. In the GBM, the volatility is assumed to be a constant. In reality, the volatility may itself be stochastic. We use GARCH(1,1) as a stochastic volatility model:

$$\sigma_t^2 = \omega + \alpha u_{t-1}^2 + \beta\sigma_{t-1}^2, \tag{7}$$

where $u_t = \ln(S_t/S_{t-1})$, and α and β are weights for u_{t-1}^2 and σ_{t-1}^2 respectively. It is required that $\alpha + \beta < 1$ for the stability of GARCH(1,1). The constant ω is related to the long term average volatility σ_L by $\omega = (1 - \alpha - \beta)\sigma_L$. The discretized version is:

$$S_{t+1} = S_t\exp\{(\mu - \sigma_t^2/2)\Delta t + \sigma_t\sqrt{\Delta t}\epsilon\}. \tag{8}$$

To estimate the parameters for the SV model in (7) and to generate sample paths, we use the MATLAB GARCH toolbox functions 'garchfit' and 'garchsim'.

4.2 Basis Functions

LSPI, FQI and LSM need to choose basis functions to approximate the expected continuation value. As suggested in [10],we use the constant $\phi_0(S) = 1$ and the following Laguerre polynomials to generalize over the stock price: $\phi_1(S) = \exp(-S'/2)$, $\phi_2(S) = \exp(-S'/2)(1 - S')$, and $\phi_3(S) = \exp(-S'/2)(1 - 2S' + S'^2/2)$. We use $S' = S/K$ instead of S in the basis functions, where K is the strike price, since the function $\exp(-S/2)$ goes to zero fast. LSPI and FQI also generalize over time t. We use the following functions for time t: $\phi_0^t(t) = \sin(-t\pi/2T + \pi/2)$, $\phi_1^t(t) = \ln(T - t)$, $\phi_2^t(t) = (t/T)^2$, guided by the observation that the optimal exercise boundary for an American put option is a monotonic increasing function, as shown in [6].

American stock put options. The intrinsic value of an American stock put options is $g(S) = \max(0, K - S)$. LSM uses the functions $\phi_0(S)$, $\phi_1(S)$, $\phi_2(S)$, and $\phi_3(S)$. LSM computes different sets of weight vectors for the basis functions for different time steps. LSPI and FQI use the functions: $\phi_0(S,t) = \phi_0(S)$, $\phi_1(S,t) = \phi_1(S)$, $\phi_2(S,t) = \phi_2(S)$, $\phi_3(S,t) = \phi_3(S)$, $\phi_4(S,t) = \phi_0^t(t)$, $\phi_5(S,t) =$

$\phi_1^t(t)$, and $\phi_6(S,t) = \phi_2^t(t)$. LSPI (FQI) determines a single weight vector over all time steps to calculate the continuation value.

American Asian call options. Asian options are exotic, path-dependent options. We consider a call option whose payoff is determined by the average price Avg of a stock over some time horizon, and the option can be exercised at any time after some initial lockout time period. The intrinsic value is $g(Avg) = \max(0, Avg - K)$. The choice of the eight basis functions for a stock price and the average of stock price follows the suggestion in [10]: a constant, the first two Laguerre polynomials for the stock price, the first two Laguerre polynomials for the average stock price, and the cross products of these Laguerre polynomials up to third order terms. LSPI and FQI take time as a component in the state space. We use the same set of basis functions for time t as those used for the American stock put options.

4.3 Results for American Put Options: Real Data

For real data, a pricing method can learn an exercise policy either 1) from sample paths generated from a simulation model; or, 2) from sample paths composed from real data directly. The testing sample paths are from real data. We scale the stock prices, so that, for each company, the initial price for each training path and each testing path is the same as the first price of the whole price series of the company.

Now we proceed with the first approach. The simulation model for the underlying stock process follows the GBM in (4) or the SV model in (7). For the GBM model, the constant volatility σ is estimated from the training data with MLE. For the SV model, we use the popular GARCH(1,1) to estimate the parameters, ω, α and β in (7). In this case, for options with quarterly, semi-annual and annual maturities respectively, the first 662, 625 and 751 stock prices are used for estimating parameters in (4) and in (7). Then LSPI, FQI and LSM learn exercise policies with 50,000 sample paths, generated using the models in (4) or in (7) with the estimated parameters. We call this approach of generating sample paths from a simulation model with parameters estimated from real data as LSPI_mle, LSPI_garch, FQI_mle, FQI_garch, LSM_mle and LSM_garch, respectively.

In the second approach, a pricing method learns the exercise policy from sample paths composed from real data directly. Due to the scarcity of real data, as there is only a single trajectory of stock price time series for each company, we construct multiple trajectories following a windowing technique. For each company, for quarterly, semi-annual, annual maturity terms, we obtain 600, 500, 500 training paths, each with *duration* = 63, 126, 252 prices. The first path is the first *duration* days of stock prices. Then we move one day ahead and obtain the second path, and so on. LSPI and LSM then learn exercise policies on these training paths. We call this approach of generating sample paths from real data directly as LSPI_data, FQI_data and LSM_data, respectively.

After the exercise policies are found by LSPI, FQI and LSM, we compare their performance on testing paths. For each company, for quarterly, semi-annual, annual maturity terms, we obtain 500, 450, 250 testing paths, each with *duration* =

63, 126, 252 prices, as follows. The first path is the last *duration* days of stock prices. Then we move one day back and obtain the second path, and so on.

For each maturity term of each of the Dow Jones 30 companies, we average payoffs over the testing paths. Then we average the average payoffs over the 30 companies. Table 1 shows the results for each company and the average over 30 companies for semi-annual maturity. Table 2 presents the average results. These results show that LSPI and FQI gain larger average payoffs than LSM.

Table 1. Payoffs of LSPI_mle, LSPI_garch, LSPI_data, FQI_mle, FQI_garch, FQI_data, LSM_mle, LSM_garch, and LSM_data, for American put stock options of Dow Jones 30 companies, with semi-annual maturity. Interest rate $r = 0.03$. 500 sample paths are composed for the discovery of exercise policies. The results are averaged over 450 testing paths.

Name	LSPI			FQI			LSM		
	mle	garch	data	mle	garch	data	mle	garch	data
3M	2.448	2.404	3.329	2.852	2.370	3.477	0.944	0.944	1.194
Alcoa	2.403	2.400	2.361	2.414	2.403	2.362	0.952	0.943	0.943
Altria	0.213	0.212	0.212	0.214	0.212	0.212	0.277	0.277	0.314
American Express	0.722	0.721	0.730	0.723	0.807	1.029	0.375	0.375	0.539
American Intl Group	4.359	4.298	4.475	4.892	5.364	5.723	1.733	1.733	2.186
AT&T	0.700	0.703	0.703	0.702	0.703	0.702	0.326	0.326	0.497
Boeing	0.118	0.117	0.112	0.117	0.118	0.117	0.324	0.274	0.274
Caterpillar	0.782	0.809	0.791	0.791	0.745	0.754	0.385	0.386	0.518
Citigroup	0.634	0.635	0.632	0.634	0.635	0.635	0.350	0.364	0.488
du Pont	2.244	2.266	2.174	2.181	2.119	2.081	0.855	0.784	0.784
Exxon Mobile	0.216	0.218	0.216	0.218	0.460	0.217	0.317	0.317	0.317
GE	0.846	0.863	0.844	0.849	0.854	0.844	0.331	0.331	0.340
GM	6.414	6.205	6.663	5.911	6.795	6.548	2.045	1.972	3.205
Hewlett-Packard	2.732	2.721	2.639	2.704	2.684	2.663	1.163	1.120	1.545
Honeywell	0.007	0.007	0.007	0.007	0.007	0.007	0.145	0.145	0.173
IBM	0.362	0.369	0.361	0.362	0.361	0.361	0.309	0.309	0.309
Intel	2.572	2.468	2.567	2.515	2.322	2.559	0.920	0.961	0.961
Johnson & Johnson	7.482	7.256	7.513	7.516	6.967	7.257	2.540	2.540	3.480
J. P. Morgan	0.818	0.820	0.818	0.817	0.818	0.816	0.366	0.366	0.366
McDonalds	1.862	1.846	1.893	1.886	1.850	1.873	0.574	0.574	0.868
Merck	0.519	0.518	0.516	0.525	0.517	0.519	0.321	0.321	0.389
Microsoft	0.312	0.308	0.309	0.326	0.309	0.309	0.230	0.230	0.326
Pfizer	1.989	1.895	1.815	1.859	3.029	1.830	1.014	1.014	1.343
Coca Cola	1.471	1.524	1.730	1.995	1.572	1.771	0.614	0.614	0.839
Home Depot	1.853	1.923	1.951	2.013	2.117	2.821	0.786	0.786	0.862
Procter & Gamble	0.372	0.377	0.825	0.434	0.367	0.389	0.280	0.280	0.280
United Technologies	2.685	2.686	2.685	2.693	2.685	2.685	0.867	0.862	1.415
Verizon	0.668	0.668	0.669	0.669	0.666	0.667	0.268	0.280	0.389
WalMart	2.611	2.612	2.597	2.691	2.597	2.650	1.030	1.030	1.314
Walt Disney	0.030	0.030	0.030	0.030	0.030	0.030	0.160	0.160	0.160
average	1.681	1.663	1.739	1.718	1.749	1.797	0.693	0.687	0.887

Table 2. Average payoffs of LSPI, FQI and LSM on real data for Dow Jones 30 companies, with quarterly, semi-annual (repeated from Table 1) and annual maturities.

maturity	LSPI			FQI			LSM		
	mle	garch	data	mle	garch	data	mle	garch	data
quarterly	1.310	1.333	1.339	1.321	1.341	1.331	0.573	0.572	0.719
semi-annual	1.681	1.663	1.739	1.718	1.749	1.797	0.693	0.687	0.887
annual	1.599	1.496	1.677	1.832	1.797	2.015	0.717	0.685	0.860

An explanation for LSPI and FQI gaining larger payoffs is that LSPI and FQI optimize weights across all time steps, whereas LSM is a value iteration procedure that makes a single backward pass through time. Thus, LSPI and FQI are able to eliminate some of the local errors.

LSM computes different sets of weights for the basis functions for different time steps; thus it generalizes over the space for asset prices. In contrast, LSPI and FQI deploy function approximation for both stock price and time, so that they generalize over both the space for asset prices and the space for time. Therefore LSM has a stronger representation than LSPI and FQI. However, LSPI and FQI outperform LSM.

4.4 Results for American Put Options: Synthetic Data

We evaluate the performance of LSPI, FQI and LSM with synthetic sample paths. The parameters for the GBM model in (4) and the SV model in (7) can either 1) be estimated from real data; or, 2) be set in some arbitrary manner. The training sample paths and the testing sample paths are generated using the same model with the same parameters.

Now we proceed with the case in which model parameters are estimated from real data. For each company, after estimating parameters for either the GBM model or the SV model from real data, we generate 50,000 sample paths with these parameters. LSPI, FQI and LSM discover the exercise policies with these sample paths. For each company, we evaluate the performance of the discovered policies on 10,000 testing paths, generated with the estimated parameters. The initial stock price in each of the sample path and each of the testing path is set as the first price in the time series of the company.

For each of the Dow Jones 30 companies, we average payoffs over 10,000 testing paths. Then we average the average payoffs over the 30 companies. The

Table 3. Average payoffs on synthetic data with parameters estimated from real data

maturity term	GBM model			SV model		
	LSPI	FQI	LSM	LSPI	FQI	LSM
quarterly	2.071	2.054	2.044	1.889	1.866	0.785
semi-annual	2.771	2.758	2.742	2.546	2.530	0.997
annual	3.615	3.645	3.580	3.286	3.311	1.241

results in Table 3 show that LSPI and FQI gain larger payoffs than LSM, both in the GBM model and in the SV model, with interest rate $r = 0.03$.

Again, an explanation for that LSPI and FQI gain larger payoffs is that LSPI and FQI optimize weights across all time steps, whereas LSM makes a single backward pass through time. LSPI and FQI follow the policy iteration approach, so that the policies they discover improve iteratively. LSM learns the policy only once in the backward-recursive approach with least squares regression.

We also vary various parameters for either the GBM or the SV model to generate synthetic sample paths. We vary the interest rate r from 0.01, 0.03 to 0.05, and set the strike price K (initial stock price) to 50. With GBM, we vary the constant volatility σ from 0.1, 0.3 to 0.5. With the SV model, we vary β from 0.2, 0.5 to 0.8, and set $\alpha = 0.96 - \beta$. We test the learned policies on testing paths generated with the same model and the same parameters. Results in Table 4 show that LSPI and FQI have similar performance as LSM in the experiments with synthetic data generated with the GBM model. Results in Table 5 show that LSPI and FQI outperform LSM in the experiments with synthetic data generated with the SV model. We believe the SV model, where volatility is stochastic, models the stock movement better than the GBM model, where the volatility is constant.

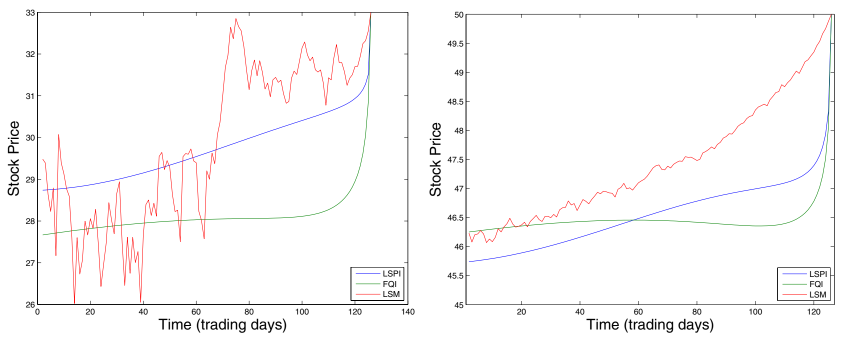
In Figure 1, we present the exercise boundaries discovered by LSPI, FQI and LSM. The optimal exercise boundary for an American put option is a monotonic increasing function, as shown in [6]. Figure 1 (a) for real data from Intel shows that the exercise boundaries discovered by LSPI and FQI are smooth and respect the monotonicity, but not the boundary discovered by LSM. The scarcity of sample paths may explain this non-monotonicity. The boundary of FQI is lower than that of LSPI, which may explain that FQI gains larger payoffs than LSPI. Figure 1 (b) shows that the exercise boundary discovered by LSPI is smoother

Table 4. Average Payoffs of LSPI, FQI and LSM. $K = 50$. Semi-annual maturity. 50,000 training paths and 10,000 testing paths are generated with the GBM model.

	$r = 0.01$			$r = 0.03$			$r = 0.05$		
σ	LSPI	FQI	LSM	LSPI	FQI	LSM	LSPI	FQI	LSM
0.1	1.294	1.300	1.286	1.095	1.117	1.061	0.925	0.902	0.896
0.3	4.086	4.062	4.095	3.684	3.666	3.679	3.533	3.604	3.504
0.5	6.965	6.798	6.051	6.514	6.521	6.476	6.315	6.274	6.365

Table 5. Average Payoffs of LSPI, FQI and LSM. $K = 50$. Semi-annual maturity. 50,000 training paths and 10,000 testing paths are generated with the SV model.

	$r = 0.01$			$r = 0.03$			$r = 0.05$		
β	LSPI	FQI	LSM	LSPI	FQI	LSM	LSPI	FQI	LSM
0.2	0.925	0.931	0.350	0.720	0.721	0.299	0.567	0.555	0.257
0.5	1.167	1.172	0.441	0.960	0.959	0.385	0.792	0.798	0.336
0.8	1.449	1.450	0.548	1.236	1.220	0.485	1.078	1.053	0.430



(a) Real data for Intel, $r = 0.03$

(b) GBM synthetic data, $r = 0.03$, 50,000 sample paths, $K = S_0 = 50$.

Fig. 1. Exercise boundaries discovered by LSPI, FQI and LSM. Semi-annual maturity.

and lower than that discovered by LSM. The exercise boundary discovered by FQI is also smooth. It crosses those of LSPI and LSM.

4.5 Results for American Asian Call Options

The experimental settings for American Asian call options are similar as those for American put options in Sections 4.3 and 4.4. For the Asian options in the experiments, there are 21 lockout days, and the average is taken over the stock prices over the last 21 days.

The experimental results comparing LSPI, FQI and LSM are shown in Table 6 to Table 9. Table 6 shows the results for the real data. Table 7 shows the results for simulation data with parameters estimated from the real data. Table 8 shows the results for simulation data generated with the GBM model. Table 9 shows

Table 6. Average payoffs of LSPI, FQI and LSM on real data. Asian options.

	LSPI			FQI			LSM		
maturity	mle	garch	data	mle	garch	data	mle	garch	data
quarterly	1.862	1.905	1.938	1.869	1.871	1.872	0.613	0.613	0.613
semi-annual	2.733	2.785	2.827	2.649	2.626	2.687	0.578	0.578	0.578
annual	4.171	4.188	4.376	4.149	4.275	4.187	0.680	0.661	0.681

Table 7. Average payoffs on simulation data with parameters estimated from real data for Dow Jones 30 companies. Asian options.

maturity	GBM model			SV model		
term	LSPI	FQI	LSM	LSPI	FQI	LSM
quarterly	2.448	2.323	0.817	2.143	2.044	0.735
semi-annual	3.951	3.897	0.822	3.573	3.377	0.738
annual	6.197	6.030	0.955	5.359	5.128	0.858

Table 8. Average Payoffs of LSPI, FQI and LSM. $K = 50$. Semi-annual maturity. 50,000 training paths and 10,000 testing paths, GBM model. Asian options.

σ	$r = 0.01$			$r = 0.03$			$r = 0.05$		
	LSPI	FQI	LSM	LSPI	FQI	LSM	LSPI	FQI	LSM
0.1	1.681	1.427	0.340	1.857	1.630	0.359	2.110	1.630	0.383
0.3	4.694	4.613	1.007	5.064	4.913	1.027	5.340	4.913	1.058
0.5	7.980	7.796	1.702	6.721	7.886	1.722	8.400	7.886	1.700

Table 9. Average Payoffs of LSPI, FQI and LSM. $K = 50$. Semi-annual maturity. 50,000 training paths and 10,000 testing paths, SV model. Asian options.

β	$r = 0.01$			$r = 0.03$			$r = 0.05$		
	LSPI	FQI	LSM	LSPI	FQI	LSM	LSPI	FQI	LSM
0.2	1.068	1.018	0.214	1.347	1.269	0.235	1.614	1.540	0.257
0.5	1.342	1.234	0.278	1.607	1.489	0.299	1.881	1.759	0.321
0.8	1.697	1.516	0.361	1.915	1.755	0.381	2.172	2.009	0.402

the results for simulation data generated with the SV model. These results show that LSPI gains similar payoffs as FQI, and both LSPI and FQI gains larger payoffs than LSM.

5 Conclusions

Options are important financial instruments, whose prices are usually determined by computational methods. Computational finance is a compelling application area for reinforcement learning research, where hard sequential decision making problems abound and have great practical significance. Our work shows that solution methods developed in reinforcement learning can advance the state of the art in an important and challenging application area, and demonstrates furthermore that computational finance remains an under-explored area for deployment of reinforcement learning methods.

We investigate LSPI for the problem of learning an exercise policy for American options, and compare it with FQI, another policy iteration method, and LSM, the standard least squares Monte Carlo method, on both real and synthetic data. The results show that the exercise policies discovered by LSPI and FQI gain larger payoffs than those discovered by LSM, on both real and synthetic data. The empirical study shows that LSPI, a solution technique from the reinforcement learning literature, as well as FQI, is superior to LSM, a standard technique from the finance literature, for pricing American options, a classical sequential decision making problem in finance.

It is desirable to theoretically analyze the policy iteration algorithm for learning an exercise policy for American options, e.g., following the approach in [1].

References

[1] Antos, A., Szepesvari, C., Munos, R.: Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. Machine Learning Journal 71, 89–129 (2008)

[2] Bertsekas, D.P.: Dynamic programming and optimal control. Athena Scientific, Massachusetts (1995)

[3] Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific, Massachusetts (1996)

[4] Bradtke, S.J., Barto, A.G.: Linear least-squares algorithms for temporal difference learning. Machine Learning 22(1-3), 33–57 (1996)

[5] Broadie, M., Detemple, J.B.: Option pricing: valuation models and applications. Management Science 50(9), 1145–1177 (2004)

[6] Duffie, D.: Dynamic asset pricing theory. Princeton University Press, Princeton (2001)

[7] Glasserman, P.: Monte Carlo Methods in Financial Engineering. Springer, New York (2004)

[8] Hull, J.C.: Options, Futures and Other Derivatives, 6th edn. Prentice Hall, Englewood Cliffs (2006)

[9] Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. The Journal of Machine Learning Research 4, 1107–1149 (2003)

[10] Longstaff, F.A., Schwartz, E.S.: Valuing American options by simulation: a simple least-squares approach. The Review of Financial Studies 14(1), 113–147 (Spring, 2001)

[11] Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks 12(4), 875–889 (2001)

[12] Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, New York (1994)

[13] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)

[14] Tsitsiklis, J.N., Van Roy, B.: Regression methods for pricing complex American-style options. IEEE Transactions on Neural Networks (special issue on computational finance) 12(4), 694–703 (2001)

Tile Coding Based on Hyperplane Tiles

Daniele Loiacono[1] and Pier Luca Lanzi[1,2]

[1] Artificial Intelligence and Robotics Laboratory (AIRLab), Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
[2] Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
{loiacono,lanzi}@elet.polimi.it

Abstract. In large and continuous state-action spaces reinforcement learning heavily relies on function approximation techniques. Tile coding is a well-known function approximator that has been successfully applied to many reinforcement learning tasks. In this paper we introduce the hyperplane tile coding, in which the usual tiles are replaced by parameterized hyperplanes that approximate the action-value function. We compared the performance of hyperplane tile coding with the usual tile coding on three well-known benchmark problems. Our results suggest that the hyperplane tiles improve the generalization capabilities of the tile coding approximator: in the hyperplane tile coding broad generalizations over the problem space result only in a soft degradation of the performance, whereas in the usual tile coding they might dramatically affect the performance.

1 Introduction

Generalization plays a key role when reinforcement learning is applied to large and complex problems and it is usually implemented by function approximation. Among the several function approximators introduced in the literature [4,8], tile coding [7] is one of the most used. Tile coding provides a piece-wise constant approximation of the target action-value function but it usually requires many tiles to accurately approximate it. In tile coding the state-action space is represented as a set of overlapping *tilings*; each *tiling* partitions the space into a set of nonoverlapping *tiles* which identify simple problem subspaces. Each tiles is associated to a weight which is updated through online experience.

In this paper we extend tile coding and we introduce the idea of *hyperplane tile coding*: the original tiles are replaced by hyperplane tiles, so that the resulting tile coding provides a piece-wise linear approximation of the action-value function, instead of the usual piece-wise constant approximation. We empirically show that hyperplane tile coding outperforms tile coding when a broad generalization on the problem space is allowed.

The paper is organized as follows. In Section 2 we briefly discuss generalization in reinforcement learning and tile coding. In Section 3 we describe in detail our extension based on hyperplane tiles. In Section 4 we describe the design of the experiments discussed in Section 5. Finally in Section 6 we draw the conclusions on our work.

2 Generalization in Reinforcement Learning

In this section we briefly introduce the notation and the basic understanding useful for the rest of this work. We assume that the reader is familiar with reinforcement learning and with tile coding (we refer the interested reader to [8] for an extensive introduction to these topics).

Reinforcement learning has proved to be effective when the action-value function is represented as a *look-up* table and each state-action pair is visited an adequate number of times. Unfortunately both these assumptions do not hold in real problems with large and often continuous state-action spaces. In fact, (i) memory requirements to store *look-up* tables quickly become infeasible; moreover, (ii) visiting adequately each state-action pair is dramatically time consuming and very often impossible. These problems raise the issue of generalization: how to represent the action-value function $Q(s, a)$ compactly while reusing collected experience in areas of the problem space scarcely or even never visited.

Generalization in reinforcement learning is usually implemented with function approximation methods: $Q(s, a)$ is represented as a function f parameterized by a vector θ which is learned from on-line experience.

2.1 Tile Coding

Several types of approximators have been introduced in the literature, e.g. RBF[3], neural networks [9]. Among the others, linear approximators [8] are the most used and seem to be the most reliables [4]. Tile coding [7] is one the most known and successfull approach to generalized reinforcement learning. It combines linear approximation with an input mapping function $\phi(s)$ that translates the state s into a vector of N binary features $\langle \phi_1(s), \dots \phi_N(s) \rangle$. Accordingly, the value of $Q(s, a)$ is computed as $\phi(s) \cdot \theta_a$, where θ_a is a vector of N parameters associated to action a.[1] In tile coding the state space is represented as a set of t overlapping *tilings*. Each *tiling* partitions the state space into a set of non-overlapping *tiles* and each tile defines a regular subspace in the state space. Given the state s, the component $\phi_i(s)$ of the features vector associated to the i-th tile is computed as,

$$\phi_i(s) = \begin{cases} 0 & \text{if } s \notin tile_i, \\ 1 & \text{if } s \in tile_i. \end{cases} \tag{1}$$

Figure 1 shows an example of a one-dimensional tile coding in which the input space is represented by four tilings, i.e. $t = 4$. In general, tilings might be placed randomly but, in practice, they are placed to cover the whole input

[1] Here we assume for simplicity that tile coding is used to generalize only over the state space and not over the action space. Several problems in the RL literature involve a small discrete action space and thus this assumption is often verified in practice; anyway all the considerations and results discussed here still hold when generalization involves also the action space.

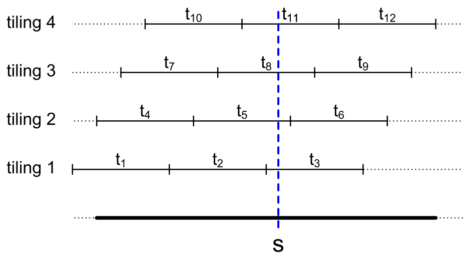


Fig. 1. Example of one dimensional tile coding

space regularly [8]: each tiling consists of tiles of width w and consecutive tilings are displaced by a resolution r as in Figure 1. Therefore, t, r and w are not three independent parameters but they are linked by the following relationship, $r = w/t$. In the following of this section we briefly discuss how the parameters $\langle r, t, w \rangle$ affects the performance of tile coding.

Resolution (r). This is probably the most important parameter. Tile coding is a piece-wise constant approximator [7], i.e. it approximate the action-value function by partitioning the state space into small regions with a constant payoff value; resolution r actually represents the width of these small regions. It is also worth noting that r completely defines the number of tiles necessary to cover the whole state space, i.e. the smaller r, the larger is the number N of tiles required; more precisely,

$$N = t \left\lceil \frac{|\mathcal{S}|}{w} \right\rceil = \left\lceil \frac{|\mathcal{S}|}{r} \right\rceil,$$

where $|\mathcal{S}|$ is the hypervolume defined by the whole state space.

Tiles Width (w). This actually defines how large are the regions of the state space associated to a single parameter and thus defines how broad generalization is. In a recent work [5] Sherstov and Stone analyzed how the parameter w affects the the learning speed. In the early stage of the learning process, large values of w result in learning very quickly a good approximation of the action-value function, whereas small values lead to very poor learning speed. On the other hand, broad generalizations (i.e. higher values of w), may be disruptive at convergence in terms of final performance [5].

Number of Tilings (t). The number of tiling t is often used to obtain the desired value of w. In fact, r should be fixed on the basis of the problem action-value function, whereas t can be accordingly fixed to achieve the desired value of w. The larger t, the broader the generalization is (remember that $w = rt$), the smaller t, the narrower generalization is. The value of t may also affect the computational complexity of the tile coding approximator: in some implementations the order of computational complexity is linear in t.

3 Extending Tile Coding with Hyperplane Tiles

Tile coding learns a piece-wise constant approximation of the action-value function; thus, it requires a small value of r in order to accurately approximate the payoff surface. Here we introduce an extension of tile coding by replacing the scalar weight associated to each tile with a parameterized hyperplane; this new tile coding version requires two main modifications and we refer to it as *hyperplane tile coding.*

First in hyperplane tile coding the weight $\theta_{a,i}$ associated to each tile is replaced by a pair $\langle \mathbf{p}_{a,i}, q_{a,i} \rangle$, where $\mathbf{p}_{a,i}$ is a parameter vector with the same numbers of variables of the state space $\mathcal{S}$ and where $q_{a,i}$ is a constant coefficient. The contribution of the i-th active tile in a given state s is no more a scalar weight $\theta_{a,i}$ (not dependent from s) but is computed as,

$$\theta_{a,i}(s) = \mathbf{p}_{a,i} \cdot \mathbf{s} + q_{a,i} \tag{2}$$

where $\mathbf{p}_{a,i} \cdot \mathbf{s}$ is a scalar vector product. The geometric interpretation of Equation 2 is a high dimensional plane, i.e., each tile represents a hyperplane approximating the action-value function. The computation of the action-value function $Q(s, a)$ does not substantially differ from standard tile coding:

$$Q(s, a) = \boldsymbol{\phi}(s) \cdot \boldsymbol{\theta}_a(s) \tag{3}$$

where $\boldsymbol{\theta}_a(s)$ is the vector $\langle \theta_{a,1}(s), \cdots, \theta_{a,N}(s) \rangle$ of the contributions of the N tiles associated to action a computed accordingly to the Equation 2.

Finally, in hyperplane tile coding, the learning of $\theta_{a,i}$ is replaced by the learning of the pairs $\langle \mathbf{p}_{a,i}, q_{a,i} \rangle$; at each time step, the parameters of each active tile are adjusted using the NLMS [2] update rule:

$$\mathbf{p}_{a,i} \leftarrow \mathbf{p}_{a,i} + \frac{\alpha}{t} \cdot \delta \cdot \frac{\mathbf{s}}{(1 + |\mathbf{s}|^2)} \tag{4}$$

$$q_{a,i} \leftarrow q_{a,i} + \frac{\alpha}{t} \cdot \delta \cdot \frac{1}{(1 + |\mathbf{s}|^2)} \tag{5}$$

where $\langle \mathbf{p}_{a,i}, q_{a,i} \rangle$ are the parameters of an active tile i in the last visited state-action pair (s, a), α is the *learning rate*, t is the tiling number, and δ is the error of the actual estimate of the action-value function $Q(s, a)$. The computation of δ actually depends on the reinforcement learning algorithm used (e.g. Q-learning SARSA, etc.). As an example, Algorithm 1 reports the pseudo code for the implementation of Q-learning based on hyperplane tile coding.

Overall hyperplane tile coding provides a piece-wise linear approximation of the payoff surface whereas standard tile coding actually provides a piece-wise constant approximator [7]. Figure 2 compares hyperplane tile coding to usual tilecoding on the approximation of a sinusoidal function; the figure shows the learned approximation after 1000 training samples (Figure 2, top row) and after 100000 samples (Figure 2, bottom row). Standard tile coding (Figure 2, left column) learns constant valued tiles (the thick solid lines) that overall results in a

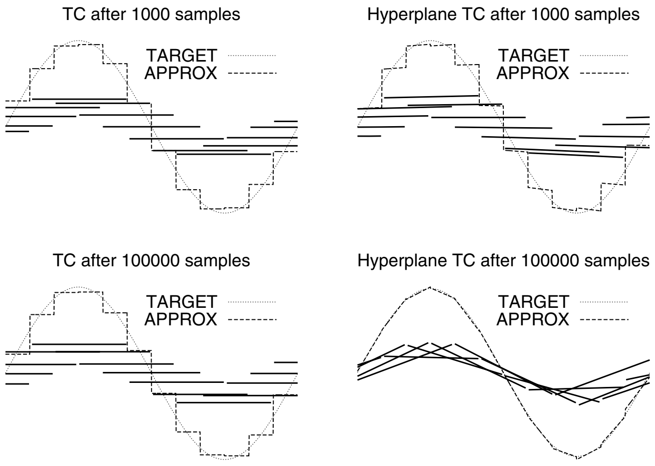


Fig. 2. Approximation capabilities of tile coding and hyperplane tile coding. Thick lines represent the contribution of every single tile.

piece-wise constant approximation as accurate as allowed by the given resolution r. Instead, hyperplane tile coding (Figure 2, right column) slowly adapts the slope of each tile to fit the slope of the target function, leading to a more accurate final approximation of the target function. Hyperplane tiles increase the number of parameters and thus the computational complexity both in terms of memory and time. On the other hand hyperplane tiles seem to result in a more accurate approximation (Figure 2) of the value function and thus may allow to use a larger value of tiling resolution r without affecting negatively the performance. Moreover the simple experiment reported in Figure 2 suggests that adapting effectively the hyperplane tiles may require a lot of training. In the next sections we present an empirical analysis of the performances of hyperplane tile coding.

4 Experimental Design

In order to test the capabilities of the hyperplane tile coding we considered three well-known problems widely studied in the reinforcement learning literature [1,7]: the *2D gridworld*, the *puddle world*, and the *mountain car*.

In this paper, we compared the performance of tile coding and hyperplane tile coding using Q-learning with a discount factor $\gamma = 1.0$ and a learning rate $\alpha = 0.2$. A run consists of a sequence of episodes; each episode starts from a random point in the state space $\mathcal{S}$ and ends when a terminal state is reached. Each episode can last at most 500 steps in the 2D gridworld and in the puddle world; in the mountain car, an episode can last for at most 5000 steps; when this limit is reached the episode stops even if the goal is not reached. In *learning episode* the actions performed are selected according to an ε-greedy exploration policy (in

Algorithm 1. Hyperplane tile coding with Q-learning

```
1: $\langle p[a][i], q[a][i] \rangle \leftarrow \langle 0, 0 \rangle$ for each action $a$ and tile $i$    ▷ Initialization
2: for all episode do
3:     $t \leftarrow 0$    ▷ Initialize the time step $t$
4:     Randomly initialize $s_t$    ▷ Initialize the problem state
5:     while (($s_t$ is not terminal) AND ($t < MaxSteps$)) do
6:         for all action $a$ do
7:             $Q[a] \leftarrow \boldsymbol{\phi}(s_t) \cdot \boldsymbol{\theta}_a(s_t)$
8:         end for
9:         if ((learning episode) AND (random() $< \varepsilon$)) then    ▷ Take a random action
10:            $a_t \leftarrow$ random action
11:        else    ▷ Take the best action according to $Q[\cdot]$
12:            $a_t \leftarrow argmax_a Q[a]$
13:        end if
14:        Take action $a_t$; observe $r_{t+1}$ and $\mathbf{s}_{t+1}$
15:        for all action $a$ do
16:            $NextQ[a] \leftarrow \boldsymbol{\phi}(s_{t+1}) \cdot \boldsymbol{\theta}_a(s_{t+1})$
17:        end for
18:        $\delta = r_{t+1} + \gamma \max_{a'}(NextQ[a']) - Q[a_t]$
19:        for all tile $i$ active in $s_t$ do
20:            $p[a_t][i] \leftarrow p[a_t][i] + \alpha\delta\mathbf{s}_t/(1 + |\mathbf{s}_t|^2)$
21:            $q[a_t][i] \leftarrow q[a_t][i] + \alpha\delta/(1 + |\mathbf{s}_t|^2)$
22:        end for
23:        $t \leftarrow t + 1$
24:    end while
25:    if (learning episode) then
26:        Set next episode as test episode
27:    else
28:        Set next episode as learning episode
29:    end if
30: end for
```

the experiments reported in this paper we always set $\varepsilon = 0.5$). In *test episodes*, the best action is always selected. Learning episodes and test episodes alternate. The learning episodes are used to explore the state-action space whereas the test episodes are used to measure the system performance. In this paper we used the same measures of performance reported by [7]: (i) in the 2D gridworld and in the mountain car, the performance is computed as the average number of steps necessary to reach the goal; (ii) in the puddle world, the performance is computed as the average cost per episode. In addition, we evaluated the performance of the learned policy at two different stages of the learning process. First, the average performance of the learned policy in the first 50 episodes is evaluated as a measure of the learning speed. Second, the performance is evaluated at the end of the learning process as a measure of the quality of the learned policy. All the statistics reported in this paper are averaged over 20 runs. In addition to the performance measures introduced above, we are also interested in measuring the number of tiles used by the approximators in each experiment.

5 Experimental Results

In the first experiment we applied tile coding and hyperplane tile coding to the 2D gridworld problem. Figure 3 compares the performance of the two approximators in the first 1000 episodes. The results show that the usual tile coding performs slightly better than hyperplane tile coding at the very beginning of the learning process. On the other hand, the hyperplane tile coding is able to reach an optimal or near-optimal performance with all the parameter settings, while the usual tile coding exhibits a poor performance when the higher value of the resolution parameter ($r = 0.04$) is used. A summary of the results, including also two additional parameter settings, is reported in Table 1. Table 1a shows the performance in the first 50 test episodes while Table 1b shows the final performance. The first column reports the parameter setting used, the second column reports the number of tiles N used; the columns TC and HTC report the performance of respectively the usual tile coding and of the hyperplane tile coding; to compare the performance of tile coding and of hyperplane tile coding we first performed a Levene's test for equal variance followed by a *t-test* for independent samples and we reported the resulting p-values in the last column of the table. The results on the first 50 test episodes (Table 1a) show, as expected, that both tile coding (column TC in Table 1a) and hyperplane tile coding (column HTC in Table 1a) perform better with high values of w: the broader are the tiles, i.e., the higher is the value of w, the faster is learning. It is also worth noting that, with the same width w, high values of r may negatively affect the learning speed: when $w = 0.16$ both tile coding (TC) and hyperplane tile coding (HTC) perform better when r is small ($r = 0.01$). The p-values in Table 1a suggest that, although the difference between the two approximators is not always significant,

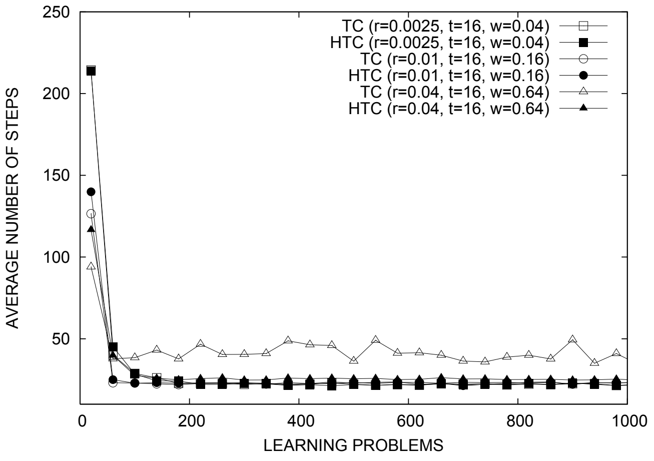


Fig. 3. Tile coding (TC) and hyperplane tile coding (HTC) applied to the 2D gridworld problem using different values of the parameters r, w and t. Curves are averages on 20 runs.

Table 1. Comparison of tile coding (TC) and hyperplane tile coding (HTC) on the 2D gridworld problem. Average number of steps to reach the goal (a) in the first 50 learning episodes and (b) after 100000 learning episodes.

$\langle r, t, w \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .0025, 16, .04 \rangle$	400	130.70 ± 12.82	131.92 ± 13.13	0.787
$\langle .01, 4, .04 \rangle$	100	154.40 ± 13.19	168.165 ± 13.8612	0.031
$\langle .01, 16, .16 \rangle$	100	74.86 ± 9.27	81.70 ± 7.77	0.003
$\langle .04, 4, .16 \rangle$	25	100.69 ± 11.83	122.07 ± 14.42	0.006
$\langle .04, 16, .64 \rangle$	25	72.00 ± 13.08	89.40 ± 17.90	0.000

(a)

$\langle r, t, w \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .0025, 16, .04 \rangle$	400	22.09 ± 0.07	21.96 ± 0.06	0.000
$\langle .01, 4, .04 \rangle$	100	22.77 ± 0.10	21.97 ± 0.05	0.000
$\langle .01, 16, .16 \rangle$	100	22.78 ± 0.08	22.01 ± 0.09	0.000
$\langle .04, 4, .16 \rangle$	25	41.06 ± 0.70	22.20 ± 0.09	0.000
$\langle .04, 16, .64 \rangle$	25	41.56 ± 0.64	22.29 ± 0.08	0.000

(b)

hyperplane tile coding appears to be usually slower than tile coding. This is not surprising because, as reported in Section 3, adapting the hyperplane tiles to fit the payoff surface may require a lot of experience. On the other hand, the final performances reported in Table 1b show that hyperplane tile outperforms the usual tile coding by exploiting its better approximation capabilities. In fact, while tile coding suffers from a dramatic decrease of performance when the number of tiles is reduced from 100 to 25, hyperplane tile coding only exhibits a soft degradation of the performance from an average of 22.01 steps to 22.20 steps. However, hyperplane tile coding requires three parameters for each tile whereas tile coding requires only one parameter for each tile, i.e., given the same number of tiles, hyperplane tile coding requires three times more the number of parameters required by standard tile coding. Anyway, even when we are fair and compare the performance of tile coding with N=100 with the performance of hyperplane tile coding with N=25, we still find that hyperplane tile coding performs significantly better than tile coding.

In the second experiments we applied tile coding and hyperplane tile coding to a slightly more complex problem, the puddle world problem. Figure 4 compares the performance, computed as the average cost per episode [7], of tile coding and hyperplane tile coding in the first 1000 episodes. The results confirm what was previously found: although the hyperplane tile coding is slightly slower than tile coding, it reaches an optimal or near-optimal performance with all the parameters settings. Table 5a reports the performance of tile coding and hyperplane tile coding in the first 50 test episodes. In this case, the disruptive effects of a higher value of r on the initial learning speed is even more evident.

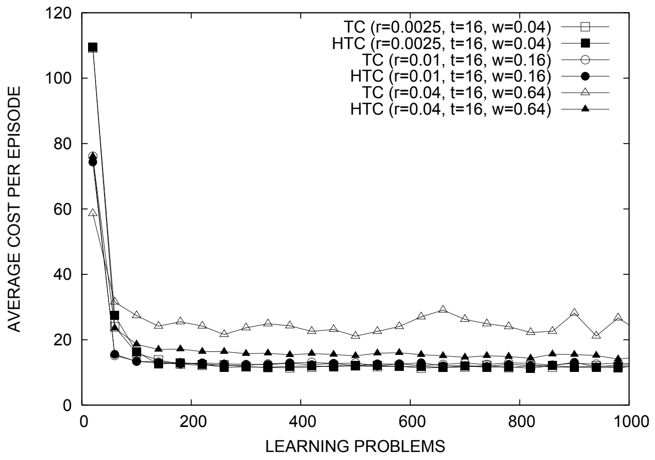


Fig. 4. Tile coding (TC) and hyperplane tile coding (HTC) applied to the puddle world problem using different values of the parameters r, w and t. Curves are averages on 20 runs.

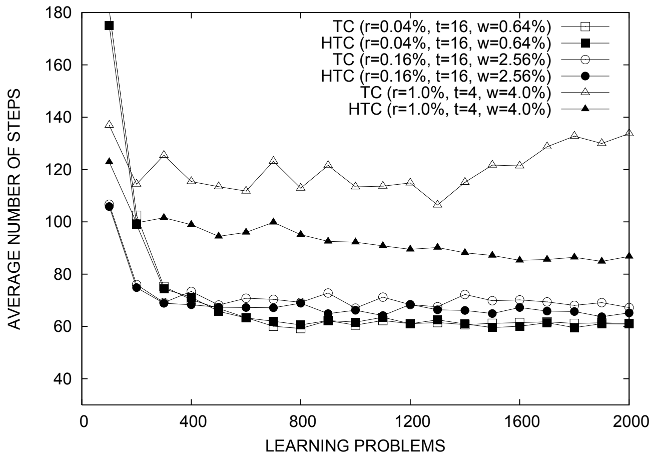


Fig. 5. Tile coding (TC) and hyperplane tile coding (HTC) applied to the mountain car problem using different values of the parameters r, w and t. Curves are averages on 20 runs.

For instance, both tile coding and hyperplane tile coding perform better when $w = 0.16$ and $r = 0.01$ than using $w = 0.64$ and $r = 0.04$. The final performances (Table 5b) suggest that hyperplane tile coding is more reliable especially when few resources, i.e., few tiles, are used.

Finally, we compared the two versions of tile coding on the mountain car problem. Figure 5 shows the performance in the first 1000 episodes. As in the previous experiments with a broad resolution the hyperplane tile coding is able to out-

Table 2. Comparison of tile coding (TC) and hyperplane tile coding (HTC) on the puddle world problem. Average number cost per episode to reach the goal (a) in the first 50 learning episodes and (b) after 100000 learning episodes.

$\langle r, t, w \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .0025, 16, .04 \rangle$	400	67.94 ± 5.00	69.12 ± 6.41	0.559
$\langle .01, 4, .04 \rangle$	100	80.99 ± 6.66	86.44 ± 7.08	0.455
$\langle .01, 16, .16 \rangle$	100	44.16 ± 6.95	45.58 ± 3.19	0.021
$\langle .04, 4, .16 \rangle$	25	61.82 ± 10.45	64.05 ± 8.72	0.029
$\langle .04, 16, .64 \rangle$	25	45.81 ± 8.455	53.77 ± 10.20	0.510

(a)

$\langle r, t, w \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .0025, 16, .04 \rangle$	400	11.66 ± 0.07	11.54 ± 0.05	0.000
$\langle .01, 4, .04 \rangle$	100	12.27 ± 0.07	11.62 ± 0.05	0.000
$\langle .01, 16, .16 \rangle$	100	12.37 ± 0.06	11.66 ± 0.05	0.000
$\langle .04, 4, .16 \rangle$	25	22.70 ± 0.32	11.99 ± 0.05	0.000
$\langle .04, 16, .64 \rangle$	25	23.23 ± 0.76	12.15 ± 0.06	0.000

(b)

Table 3. Comparison of tile coding (TC) and hyperplane tile coding (HTC) on the mountain car problem. Average number of steps per episode to reach the goal (a) in the first 50 learning episodes and (b) after 250000 learning episodes. The value $r\%$ and $w\%$ are respectively $r/|S|$ and $w/|S|$.

$\langle r\%, t, w\% \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .01, 64, .64 \rangle$	10000	216.57 ± 34.06	224.25 ± 52.68	0.902
$\langle .04, 16, .64 \rangle$	2500	146.79 ± 33.11	160.00 ± 31.16	0.168
$\langle .16, 16, 2.56 \rangle$	625	241.56 ± 41.89	243.38 ± 40.97	0.355
$\langle 1., 4, 4. \rangle$	100	182.62 ± 51.24	167.11 ± 42.18	0.620

(a)

$\langle r\%, t, w\% \rangle$	N	TC ($\mu \pm \sigma$)	HTC ($\mu \pm \sigma$)	p-value
$\langle .01, 64, .64 \rangle$	10000	56.58 ± 0.33	56.12 ± 0.36	0.000
$\langle .04, 16, .64 \rangle$	2500	68.17 ± 0.43	58.95 ± 0.46	0.000
$\langle .16, 16, 2.56 \rangle$	625	60.23 ± 0.34	57.63 ± 0.35	0.000
$\langle 1., 4, 4. \rangle$	100	122.95 ± 2.43	61.20 ± 0.32	0.000

(b)

perform the usual tile coding, achieving a good tradeoff between the number of parameters used and the final performance. In the first 50 test episodes (Table 3a) hyperplane tile coding always performs worse than tile coding, nevertheless such difference is not always statistically significant. However, the final performances show that hyperplane tile coding always perform significantly better than tile coding and, as usual, it provides a soft degradation of the performance when the number of tiles is dramatically reduced.

6 Conclusions

We have introduced hyperplane tile coding, an extension of tile coding in which the usual tiles are replaced by hyperplanes: the original weight associated to each tile is thus replaced by a parameterized hyperplane. The proposed extension leads to an increase of the computational complexity, both in terms of memory and time, that is linear in the number of problem variables. On the other hand, the use of hyperplanes instead of the usual piece-wise constant ones seems to require fewer tiles to provide an accurate approximation of the action-value function. Therefore, although theoretically more expensive, hyperplane tile coding might result in practice cheaper than standard tile coding. We compared empirically the standard tile coding with the hyperplane tile coding on three well-known benchmark problems. Our results suggest that hyperplane tile coding (i) provides robust performance when the number of parameters is reduced; (ii) can perform slightly (though significantly) better than tile coding with less memory requirements. Although the results obtained are still preliminary, we think that hyperplane tile coding might be a promising alternative to the standard tile coding. Future work will include the extension of the hyperplane tile coding with more effective update algorithms such as K1 or IDBD [6].

References

1. Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 369–376. The MIT Press, Cambridge (1995)
2. Haykin, S.: Adaptive Filter Theory. Prentice-Hall information and system sciences series (2002)
3. Kretchmar, R., Anderson, C.: Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In: Proceedings of the IEEE International Conference on Neural Networks, Houston, TX, pp. 834–837 (1997)
4. Reynolds, S.I.: Reinforcement Learning with Exploration. Ph.D thesis, School of Computer Science. The University of Birmingham, Birmingham, B15 2TT (December 2002)
5. Sherstov, A.A., Stone, P.: Function approximation via tile coding: Automating parameter choice. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS, vol. 3607, pp. 194–205. Springer, Heidelberg (2005)
6. Sutton, R.S.: Gain adaptation beats least squares? In: Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems, pp. 161–166. Yale University, New Haven (1992)

7. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems, vol. 8, pp. 1038–1044. The MIT Press, Cambridge (1996)
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning – An Introduction. MIT Press, Cambridge (1998)
9. Tesauro, G.: TD-gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation 6(2), 215–219 (1994)

Use of Reinforcement Learning in Two Real Applications[⋆]

José D. Martín-Guerrero, Emilio Soria-Olivas, Marcelino Martínez-Sober, Antonio J. Serrrano-López, Rafael Magdalena-Benedito, and Juan Gómez-Sanchis

Intelligent Data Analysis Laboratory, Department of Electronic Engineering
University of Valencia, Spain
{jose.d.martin,idal}@uv.es

Abstract. In this paper, we present two sucessful applications of Reinforcement Learning (RL) in real life. First, the optimization of anemia management in patients undergoing Chronic Renal Failure is presented. The aim is to individualize the treatment (Erythropoietin dosages) in order to stabilize patients within a targeted range of Hemoglobin (Hb). Results show that the use of RL increases the ratio of patients within the desired range of Hb. Thus, patients' quality of life is increased, and additionally, Health Care System reduces its expenses in anemia management. Second, RL is applied to modify a marketing campaign in order to maximize long-term profits. RL obtains an individualized policy depending on customer characteristics that increases long-term profits at the end of the campaign. Results in both problems show the robustness of the obtained policies and suggest their use in other real-life problems.

1 Introduction

Reinforcement Learning (RL) has been widely studied theoretically. There are also many known applications, especially in Robotics. However, its use to solve problems beyond Robotics is still scarce. This might be due to the need of large databases to have meaningful state/action spaces. However, there are many real problems that can be characterized to be tackled using RL. In this work, we focus on two real applications of RL:

1. *Optimization of anemia management in patients undergoing hemodialysis.* This is a relevant problem in Nephrology, in which we focus on obtaining the optimal Erythropoietin (EPO) dosages that should be administered for an adequate long-term anemia management.
2. *Optimization of a marketing campaign.* In this case, we used data from a marketing campaign to suggest modifications based on RL to the company policy in order to maximize long-term profits.

The organization of the paper is as follows. Section 2 is devoted to the problem of optimization of anemia management. Section 3 focuses on the problem of optimization of a marketing campaign. Finally, Section 4 gives some concluding remarks and suggestions for future work.

[⋆] This work has been partially supported by the research projects ARVIV/2007/094, CSD2007-00018 and TIN2007-61006.

2 Optimization of EPO Dosages in Patients Undergoing Chronic Renal Failure

2.1 Description of the Problem

Anemia is a nearly universal sequel in an End-Stage Renal Disease (ESRD) patient. Until the advent of EPO, ESRD patients faced severe anemia and cardiovascular complications, or the requirement for multiple transfusions. The addition of this expensive drug to the already burdensome cost of the Medicare ESRD program involves considerable expenses for the Health Care System (HCS). It is crucial to make a good use of this drug by means of personalized dosages and number of administrations. This will guarantee an adequate pharmacotherapy as well as a reasonable cost of the treatment.

Anemia treatment has two stages: correction and maintenance. In the correction stage, an Erithropoyetic Stimulating Factor (ESF) is used since it is recommended to increase the Hemoglobin (Hb) level within 4–8 weeks. In the maintenance stage, and depending on the patient's response, some changes may be done either in dosages or weekly number of administrations. It is important to point out that there is an important risk of side effects associated with ESFs if Hb levels are too high or they increase too fast. These side effects are basically related to trombo embolisms and vascular problems [1]. Oftentimes, the relationship between the drug dose and the patient's response is complex. To facilitate drug administration, practitioners attempt to use protocols. Such protocols are developed from average responses to treatment in populations of patients. Nevertheless, achieving a desired therapeutic response on an individual basis is complicated by the differences within the population.

The Dialysis Outcomes Quality Initiative of the National Kidney Foundation recommended Hb maintained within the narrow range of $11-12$ g/dl [2], although nowadays the most accepted recommendation by nephrologists and pharmacists is to maintain Hb within 11 and 13 g/dl [3]. In this study, our target is to maintain Hb levels within 11.5 and 12.5 g/dl. This narrower range allows to increase the sensitivity of alert criteria.

Several attempts at the automation of the EPO delivery have already been reported based on parametric identification in a Bayesian framework [4], on a fuzzy rule-based control strategy [5], and on Artificial Neural Networks (ANNs) [6,7]. All these works are based on approximators or predictors that can obtain the Hb concentration or the optimal EPO dosage for the next control in order to attain a certain Hb concentration. This is an interesting approach but it shows a major flaw, namely, this kind of predictors optimize the output variable a pre-defined number of steps ahead, but it is extremely difficult to obtain long-term predictors because the models are restrictive in terms of both the number of considered delays and steps ahead.

What makes RL a suitable technique for this problem is its way of tackling the problem as achieving long-term stability in patients' Hb level. This processing is much closer to medical reasoning than the processing followed by any of the other approaches proposed previously (Bayesian theory, fuzzy logic, ANNs, etc.). This is because RL finds a suitable policy, i.e., given a patient in a certain state, RL provides the sequence of actions that will lead the patient to the best possible state. The goodness of the state is appropriately defined by means of the rewards assigned to the different possible values of Hb levels.

There are two basic RL approaches: on-policy and off-policy [8]. An on-policy RL method modifies the starting policy towards the optimal one. In our particular application, patients would be probed by possibly non-optimal policies during an episodic learning process. Construction of such a policy requires sufficiently many occurrences of all possible state transitions, potentially causing over-dosing or under-dosing. As a result, on-policy episodic RL tools can discover a useful dosing policy, as a product of a learning process, which may be however unacceptably long and dangerous in real-time pharmacotherapy.

In an off-policy approach, the optimal policy can be computed while the agent is interacting with the environment by means of another arbitrary policy. In this work, we use the most widely known off-policy RL method (Q-learning). Therefore, our goal is to stabilize the Hb level within the target range of $11.5 - 12.5$ g/dl. The Q-learning mechanism avoids probing the system by suboptimal dosing policies during long training episodes for evaluation of the state/action pairs. The proposed learning system determines the optimal drug dose using reinforcements, which are produced immediately after state transitions occurring within the patient dynamics during treatment [9,10].

2.2 Data Collection

Patients with secondary anemia due to Chronic Renal Failure (CRF) in periodic hemodyalisis were included in this study. All patients were treated in the University Hospital Dr. Peset (Valencia, Spain). Patients were monitored monthly. We used two sets of patients: a cohort of 64 patients treated during 2005 and 77 patients analyzed during 2006. Several factors for each patient were usually collected in their follow-up: plasmatic concentration of Hb (g/dl), Hematocrit concentration (%), ferritin (mg/l), administration of Intra-Venous Iron (IV Fe) (mg/month), number of administrations of IV Fe, weekly dosage of EPO beta (International Units, IU), weekly dosage of darbepoetin alpha (μg), number of administrations of these ESFs, and other variables that the Pharmacy Unit staff considered irrelevant for our study, namely, age, weight, sex, Calcium (mg/l), Phosporus (mg/l), Phosphate (IU), PTH-I (pg/ml), dosage of Calcitriol and number of administrations of Calcitriol.

2.3 Experimental Setup

In order to accommodate our data to RL algorithms, we considered each monthly test as an episode. After a preprocessing stage, seven variables were selected to define the state space: Hb level, ferritin, dosage of IV Fe, dosage of EPO beta, dosage of Darbepoietin alpha and number of administration of these ESFs.

Percentile analysis as well as expert advice were used to define the most adequate thresholds for the ranges used to discretize the variables. In particular, the variable Hb was divided into seven ranges[1], ferritin into five ranges, IV Fe dose into four ranges,

[1] These seven ranges were the same as those used to select the different values of rewards. These values were selected by the nephrologists due to their medical relevance.

EPO beta into five ranges, number of administrations of EPO beta into five ranges, and finally, both Darbepoietin alpha and number of administrations of Darbepoietin alpha into four ranges.

Nine actions were chosen according to nephrologists: increase/decrease EPO beta dosage; increase/decrease Darbepoietin alpha dosage; increase/decrease number of administrations of EPO beta; increase/decrease number of administrations of Darbepoietin alpha; maintain dosage and number of administrations with no changes. Since patients were treated using either one kind of EPO or the other, there was not any action involving changes in the kind of EPO. Rewards depended on the Hb level, since the final goal was to maintain patients within 11.5 g/dl and 12.5 g/dl. Rewards were defined according to experts' advice, as shown in Eq. (1).

$$\text{reward} = \left\{ \begin{array}{ll} 10, & if\ Hb \in [11.5 - 12.5] g/dl \\ \\ 0, & \begin{array}{l} if\ Hb \in [10.0 - 11.5] g/dl \\ or\ Hb \in [12.5 - 13.0] g/dl \end{array} \\ \\ -5, & \begin{array}{l} if\ Hb \in [9.0 - 10.0] g/dl \\ or\ Hb \in [13.0 - 14.0] g/dl \end{array} \\ \\ -10, & otherwise \end{array} \right\} \tag{1}$$

We used an ANN, such as the Multilayer Perceptron (MLP), as function approximator [11]. The inputs to the MLP are given by the variables that define the state and also by the taken action. The desired output of the MLP is given by Eq. (2). Once the model is trained, it can approximate the reward associated to any combination of states and actions. The use of this function approximator has two basic approaches; the first one is based on swapping the tabular Q-learning algorithm with the function approximator in all cases; and the second one, is based on swapping only if the action-state combination has not yet been previously experienced. The latter makes more sense, since it only uses the approximator when it is impossible to obtain a policy by using tabular methods.

$$target = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) \tag{2}$$

2.4 Results

Using data from 2005, we obtained a policy that turned out to be considerably better than that used in the hospital in terms of the percentage of patients that were within the targeted range of Hb. Evaluation was carried out off-line using historic data. The best results were obtained using the tabular Q-learning algorithm for already experienced situations and the function approximator for those cases that had not yet been previously experienced. In particular, being N_m the number of monitorings and I the best value of the Q function at the last episode, i.e., the value associated to that action which guarantees the maximum value of the Q-function (greedy approach), the hospital policy evaluated by the SARSA algorithm showed a value of $\sum_{i=1}^{N_m} I_h(i) = 2359.7$, whereas the RL policy had a value of $\sum_{i=1}^{N_m} I_q(i) = 2922.0$, which involved an increase of 19%.

Our proposed policy guaranteed that Hb levels of patients were closer to the desired range of Hb than using the hospital protocol. In particular, analyzing the long-term results after one year of treatment, there was an increase of 25% in the number of patients that were within the desired range of Hb or in those ranges that were immediately next to the desired one. This ensured stable conditions in patients undergoing CRF and maximized patients' quality-of-life. Moreover, the proposed policy might involve a considerable economic saving for the hospital. The percentage of patients undergoing CRF represents between 0.1% and 0.2% of the population. The investment in ESFs of the Pharmacy Unit (PU) of the hospital analyzed in this study is around €30,000 per year. This PU has estimated than an optimal policy would save between €100 and €200 per patient and year, which would involve a rough saving of €1,000,000 for all Valencian Region[2].

The application of the previously obtained policy to the cohort of patients treated during 2006, showed an improvement in the value of $\sum_{i=1}^{N_m} I(i)$ equal to 15%. Therefore, it involved that a higher percentage of patients were within the desired range of Hb (or at least, closer to the desired range), when using the RL policy than following the hospital policy. In particular, there was an increase of 22% in the number of patients that were within the desired range of Hb at the end of year 2006. It guarantees a better Quality-of-Life, and it also involves considerable economic savings for the HCS.

Therefore, the achieved results are promising, since they indicate that the RL policy is robust and generalizable to patients different from those used to obtain the policy. To end up all the validation process, some measures should still be carried out, e.g., a complete calculation of the economic savings involved (taking into account all the factors, such, as potential admissions in hospitals, expenses in drugs related to the treatment and the state of the disease, ...). It would also be very interesting to analyze the monthly evolution of the ratio of patients that are within the desired range of Hb; this would help answer questions like: How long does it take for a certain patient to reach this desired target? Are they stable?, i.e., once they are within the desired target, what is the probability to be out of the range again in the future? Is the evolution slow and progressive? The answers of these questions could be used together with experts' advice to improve the definition of the rewards.

3 Optimization of a Marketing Campaign

3.1 Description of the Problem

The latest marketing trends are more concerned about maintaining current customers and optimizing their behavior than getting new ones. For this reason, relational marketing focuses on what a company must do to achieve this objective [12]. The relationships between a company and its customers follow a sequence of action-response system,

[2] Region of Valencia is a Spanish region with approximately 5 millions of inhabitants and an autonomous government, which manages its own Health Care System.

where the customers can modify their behavior in accordance with the marketing actions developed by the company.

One way to increase the loyalty of customers is by offering them the opportunity to obtain some gifts as the result of their purchases from the company. The company can give *virtual credits* to anyone who buys certain articles, typically those that the company is interested in promoting. After a certain number of purchases, the customers can exchange their virtual credits for the gifts offered by the company.

The problem is to establish the appropriate number of virtual credits for each promoted item and for each individual customer. In accordance with the company policy, it is expected that the higher the credit assignment, the higher the amount of purchases. However, the company's profits are lower since the marketing campaign adds an extra cost to the company. The goal is to achieve a trade-off by establishing an optimal policy. We propose the use of RL to solve this task since previous applications have demonstrated its suitability in this area [13,14], in particular to solve the mailing problem.

The main difference between the mailing problem and the credit assignment problem is that the action space becomes multi-valued instead of binary. In the mailing problem only two actions can be considered: to send a catalog or not to send a catalog. In a credit assignment application, the optimal policy should recommend how many credits should be assigned to each transaction.

Marketing problems tend to have a very complex characterization of the transactions that are involved. This highly dimensional attribute requires the implementation of RL algorithms by means of state aggregators or function regressors, which under certain conditions may lead to convergence problems [8]. In this work, we propose two different approaches. First, the state space is clustered by using a vectorial quantization carried out by algorithms based on the Self-Organizing Map (SOM); this approach enables us to work with RL tabular methods. Second, an MLP is used to predict the response of customers when different actions are carried out by the company. This prediction is then used to obtain an optimal policy.

3.2 Data Collection

Data were collected from a company[3] that was interested in designing a campaign to encourage their clients to buy more of their products. Data involved 1,264,862 transactions, 1,004 different articles, 3,573 customers and 5 (monthly) episodes. This marketing campaign was based on assigning virtual credits to customers. That assignment was carried out manually based on internal criteria.

The information used for this study corresponds to the first five months of the campaign. We have just received an extended data set formed by 41 monthly episodes that will be used to find more suitable and robust policies. Although a confidentiality agreement prevents a number of details of the campaign from being released, the main characteristics of the campaign can be made public:

1. The company assigned virtual credits to customers according to the items bought by them. When customers had enough credits, they could exchange their credits for gifts.

[3] The name of the company cannot be made public due to a confidentiality agreement.

2. Customers could obtain virtual credits by buying specific items which were indicated as “encouraged”. The company selected these promoted items monthly (according to internal criteria).
3. Since the assignment of these virtual credits involved a cost to the company, immediate profits decreased as a direct consequence of the campaign.

The credit assignment took place at the end of every episode and was computed by taking into account how many “encouraged” articles were bought by customers during that month. The so-called Life-Time Value (LTV) at time t (reward) for a certain customer was obtained as follows:

$$LTV(t) = \sum_i P_i(t) \cdot A_i(t) - K_C V_C(t) \tag{3}$$

where A_i is the amount of type i articles purchased by the customer; P_i is the price of type i articles; V_C is the number of virtual credits assigned to the customer; and K_C is a coefficient that reports the costs incurred by the company due to credits. The aim of this work is to increase LTV for every customer by using RL as the strategy to achieve an optimal policy.

Since it was not possible to carry out the improvement of the policy on-line, a batch method was used. Episodes were repeatedly shown to the RL algorithm until the convergence of the policy was achieved.

3.3 Experimental Setup

Action and State Spaces. The first task to tackle in an RL algorithm is the design of state and action spaces. This requires an exhaustive analysis of both the clients and their actions. The vast amount of information stored by the company showed that there were many features that defined the customer behavior. Specifically, the following features were included in the study:

1. The identification of the shop that sold the products.
2. The geographical area where the client made the purchase.
3. The date of the purchase transaction.
4. The number of items purchased by the client.
5. Family[4].
6. The item identification number.
7. Whether the items were regular items or “encouraged” items.
8. Price of the item purchased.

An initial classical data mining study was carried out to analyze the data. This study showed that neither geographical nor temporal information were relevant; therefore, this information was removed from further analyses.

Marketing studies consider that an optimal set of features to profile the future behavior of a customer is given by the so-called RFM variables [15]:

[4] Articles were grouped into families. A family was a label which gathered similar products.

- **Recency (R):** the most recent date when the customer made a transaction (usually a purchase, but it can also be a refund request, for instance).
- **Frequency (F):** the number of times the customer made a purchase.
- **Monetary (M):** the monetary quantity of products purchased by the customer.

It has been confirmed experimentally that the most valuable customers have high frequency and monetary values, and low recency values [15]. Although this set of features is the most suitable one to define the state space, we avoided using the same information in the definition of the state space and in the computation of the long-term reward. Therefore, the monetary feature was split into two different ones:

- **Amount:** the number of items purchased in an episode by each customer.
- **Average price:** the average price of all the purchases in an episode.

In addition, another feature was considered in the state space. Since customers obtained virtual credits by purchasing some items marked as "encouraged", it made sense to add a feature that showed how many "encouraged" items were bought by customers.

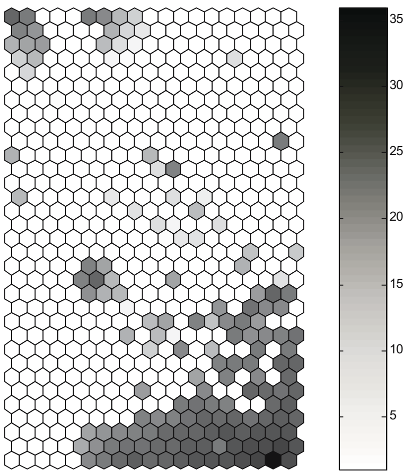
The action to be taken was the assignment of a certain number of virtual credits to customers depending on their purchases. Since there was a wide range of credits, their quantization was required to have a manageable number of possible actions. This quantization procedure was carried out taking into account the different gift values for a certain number of credits. The company established 10 categories of gifts according to their price, and hence, the action space was also divided into 10 categories.

The SOM Approach. A SOM was trained with all the input patterns from the data collection, and then, the most representative neuron for a particular input vector was computed. That neuron is usually called Best Matching Unit (BMU), and in our case, represents the state of the customer. This approach located similar customers in the same state; i.e., an aggregation of states was obtained. Given an input vector $s_t = s_t(R, F, A, A-P, E)$[5], its BMU was computed. This value was used as an entry for the Q-table. Therefore, the input patterns actually contained information about the marketing-oriented features.

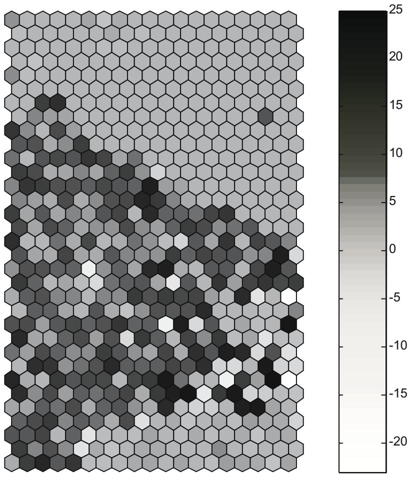
An additional advantage of this proposal is that the interpretation of results was quite straightforward due to the intuitive maps provided by the SOM. Moreover, there was no need to use function regressors, thereby avoiding the danger of non-convergent policies.

The MLP Approach. The input space was made up of the set of transaction features, and the output space consisted of the long-term rewards. Moreover, in the prediction problem, it was necessary to include not only the state features, but also the features of the actions taken by the company. It was then feasible to predict the Q-values when taking different actions in the same state. The best MLP model had one hidden layer made up by eight neurons. This architecture was obtained by a cross-validation procedures, after testing different architectures. The objectives of the training were computed using MC methods because they ensured convergence of the regression algorithm.

[5] $R \equiv$ recency, $F \equiv$ frequency, $A \equiv$ the amount of items purchased, $A - P$ is the average price, and E indicates whether or not the products are "encouraged".



(a)



(b)

Fig. 1. Policy followed by the company (a) in terms of the number of assigned credits and improvements suggested by the SOM-RL algorithm (b) in terms of the difference of assigned credits with respect to the company policy. In the company policy map, (a), the darkest area depicts a high number of credits given to customers; the medium gray area shows intermediate numbers, and the white area indicates that no credits were given to customers. In the comparison map, (b), the darkest areas mean that an increase in the credit assignment is recommended by the optimal policy; medium gray areas mean no modification in the company policy is suggested and the light gray areas (very small) indicate that a decrease in the credit assignment should be made.

3.4 Results

Since the algorithms used in this work were off-policy, the data had to be arranged in episodes and the evaluation was carried out off-line. Each episode was made up of the transactions of each customer in each one of the five temporal (monthly) steps that appeared in the data set.

SOM Results. Fig. 1 (a) shows that the policy followed by the company was very deterministic. Three zones are well defined. The darkest area depicts high number of credits given to customers; the gray area depicts intermediate amounts, and the white area indicates that no credits were given to customers. This guideline of the company policy is a disadvantage for the RL algorithms since they need all actions to be taken in all states. In spite of this lack of exploration, the RL algorithm was able to find a good enough solution for this problem. In fact, when computing the $Q(s,a)$ function by means of the SOM approach, several modifications of the actual policy were suggested as shown in Fig. 1 (b). There are two main parts in this comparison map: the upper-right and the lower-right areas (in which no modifications were suggested), and the central area where the optimal policy suggested an increase in the number of credits given to customers. The former represents well-defined customers ("loyal" customers in the case of the lower-right area and "negligible" customers for the company in the case of the upper-right area). The central area represents "average" customers.

MLP Results. The data was split into three data sets: a training data set formed by 66.6% of the patterns to train the network; a validation data set (33.3% of the patterns) to carry out a cross-validation; and finally, a test data set that consisted of the remaining 33.3% of the patterns, which had not yet been seen by the network. Unbiased models with good generalization capabilities were obtained in this way. It should be pointed out that cross-validation procedures were applied to estimated values, not to policies.

Very similar values of the Mean-Square Error (MSE) were obtained in these three data sets. In particular, errors showed low values, which corresponded to an accurate regression. Once the MLP was trained, it was used to estimate the optimal policy. The results achieved with this algorithm were difficult to visualize because the input space had 11 dimensions.

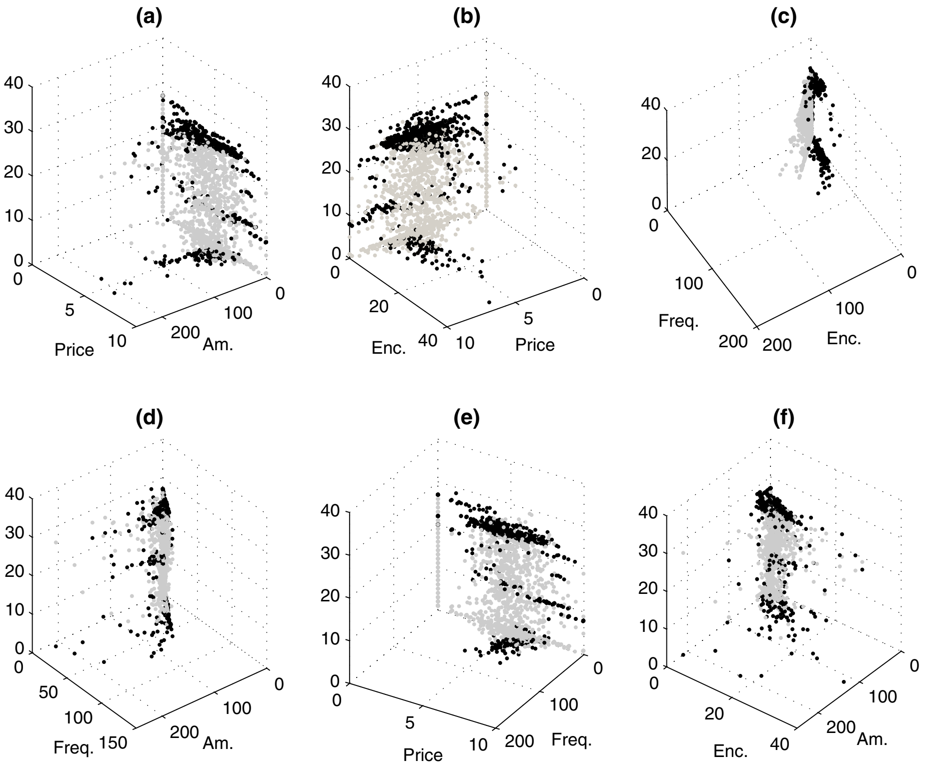


Fig. 2. Several plots showing the comparison of the optimal policy and the company policy with the features of the state space. The gray dots represent the company policy, while the black dots represent the optimal policy. The xy-plane is defined by sets of two features, while z-axis represents the number of given credits.

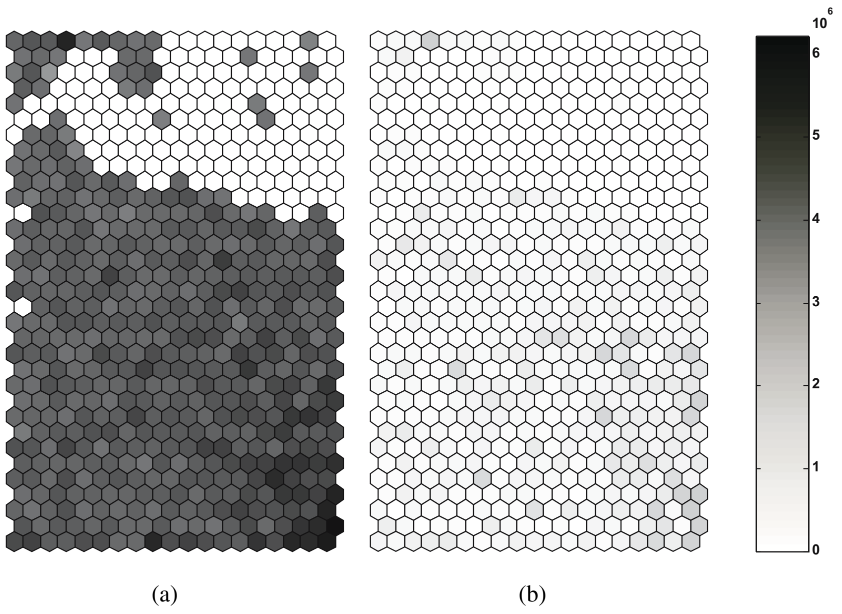


Fig. 3. Q-function in a SOM topology for the optimal policy (a) and for the company policy (b). The darker the color, the higher the value of the action-value function $Q^{\pi}(s, a)$. The RL policy provides much larger profits than the company policy. The gray-shade bar shows the value of $Q^{\pi}(s, a)$ for the different gray shades.

Fig. 2 shows several plots comparing the optimal policy and the company policy with the features that characterize states. Note that in Fig. 2 (c) (which shows both policies versus the characteristics "Frequency" and "Encouraged"), the company policy assigned virtual credits to those customers who were prone to buying "encouraged" articles. The optimal policy suggested that credits should be assigned to customers who purchased products more frequently even when these products were not "encouraged".

The overall behavior of the optimal policy was more "aggressive" than the company policy. For instance, in the company policy, actions were spread over the entire action space, whereas the two main groups in the optimal policy were located at the top and at the bottom of the action space. Similar states seemed to lead to opposite actions. The explanation of this behavior might be related to the features used to define the state space. The fact that opposing actions were recommended in similar states suggests that these features might be an incomplete state representation.

Fig. 3 shows how the optimal policy helps the company to increase its profits considerably. In those areas in which improvements were suggested, the Q-values for the optimal policy were four or even five times larger than the Q-values of the company policy.

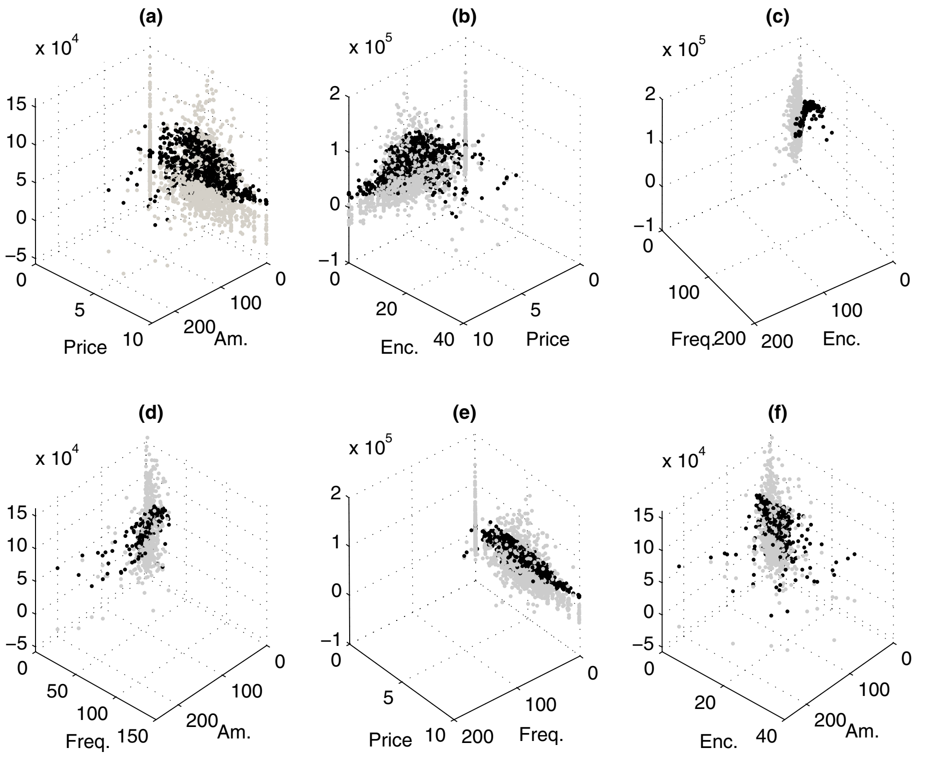


Fig. 4. Q-function for the MLP approach. Gray dots stand for the profits expected if the company policy is followed, whereas black dots correspond to the RL policy values. The xy-plane is defined by sets of two features, whereas z-axis represents the value of the Q-function.

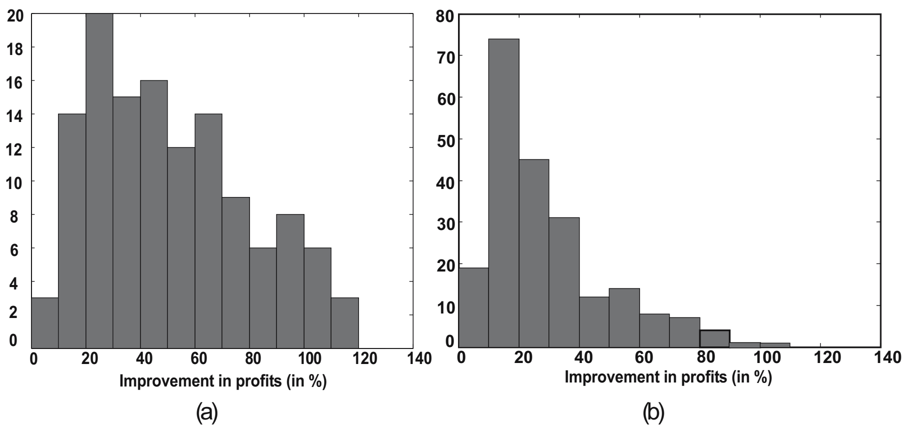


Fig. 5. Histograms showing relative profit increase when using the RL policy instead of the company policy. Non-VIP customers are shown in (a) and VIP customers in (b).

The results yielded by the MLP approach are shown in Fig. 4. The black dots are more uniformly distributed than the gray dots, which might be due to the fact that the optimal policy tried to make customers loyal to the company by ensuring a certain minimum number of sales. The behavior of the company policy was either excellent or very poor. The Q-values obtained by the RL policy were not as high as some of the results obtained by the company policy in some cases. Nonetheless, the RL policy was less likely to obtain poor results than the company policy.

Fig. 5 shows the relative profit increase following the RL policy instead of the company policy. VIP customers refer to a selected group of customers that has a special relationship with the company. An RL policy provided much higher profits than the company policy when it was evaluated using historic data. Although it is a promising result, a validation with new data is required to claim that RL can indeed get higher profits.

4 Conclusions

We have presented a work focused on the application of RL to two real problems. In the problem of optimization of anemia management, we have found policies to maximize the ratio of patients that are within a targeted range of Hb. It is remarkable the novelty of an RL application to Pharmacy, which is a field of knowledge very different to the typical ones that are related to RL. The proposed approach is completely general and can be applied to any problem of drug dosage optimization.

Moreover, two different RL approaches (state aggregation by SOM and action-value function approximation by MLP) in targeted marketing have been developed and benchmarked. The application of both algorithms has proven to be appropriate for problems of this kind because promising results have been achieved. The SOM approach is more straightforward to understand than the MLP approach due to the properties of the SOM algorithm, which allows an intuitive representation of the results. The results suggest that the use of SOM could be recommended to improve a marketing campaign. The MLP regressor is more powerful because the generalization process is more robust than in the SOM algorithm, but is is more difficult to extract information from the achieved modeling.

Our ongoing research on both problems is focused on improving and validating the policies since we have just had new data avaliable for both problems.

References

1. Lynne Peterson, L.: FDA Oncologic Drugs Advisory Committee (ODAC) meeting on the safety of erythropoietin in oncology. Trends in Medicine, pp. 1–4 (May 2004)
2. National Kidney Foundation, K.D.O.Q.I.: Guidelines for anemia of chronic kidney disease. NKF K/DOQI Guidelines (2000), http://www.kidney.org
3. Steensma, D., Molina, R., Sloan, J., Nikcevich, D., Schaefer, P., Rowland, K.J., Dentchev, T., Novotny, P., Tschetter, L., Alberts, S., Hogan, T., Law, A., Loprinzi, C.L.: Phase III study of two different dosing schedules of erythropoietin in anemic patients with cancer. Journal of Clinical Oncology 24(7), 1079–1089 (2006)

4. Bellazzi, R.: Drug delivery optimization through bayesian networks: an application to erythropoietin therapy in uremic anemia. Computers and Biomedical Research 26, 274–293 (1992)
5. Bellazzi, R., Siviero, C., Bellazzi, R.: Mathematical modeling of erythropoietin therapy in uremic anemia. Does it improve cost-effectiveness? Haematologica 79, 154–164 (1994)
6. Jacobs, A.A., Lada, P., Zurada, J.M., Brier, M.E., Aronoff, G.: Predictors of hematocrit in hemodialysis patients as determined by artificial neural networks. Journal of American Nephrology 12, 387A (2001)
7. Martín, J.D., Soria, E., Camps, G., Serrano, A., Pérez, J., Jiménez, N.: Use of neural networks for dosage individualisation of erythropoietin in patients with secondary anemia to chronic renal failure. Computers in Biology and Medicine 33(4), 361–373 (2003)
8. Sutton, R., Barto, A.: Reinforcement Learning: An Introducion. MIT Press, Cambridge (1998)
9. Martín, J.D., Soria, E., Chorro, V., Climente, M., Jiménez, N.V.: Reinforcement learning for anemia management in hemodialysis patients treated with erythropoietic stimulating factors. In: European Conference on Artificial Intelligence 2006, Proceedings of the Workshop Planning, Learning and Monitoring with uncertainty and dynamic worlds, Riva del Garda, Italy, pp. 19–24 (2006)
10. Martín, J.D., Soria, E., Martínez, M., Climente, M., De Diego, T., Jiménez, N.V.: Validation of a reinforcement learning policy for dosage optimization of erythropoietin. In: Orgun, M.A., Thornton, J. (eds.) AI 2007. LNCS, vol. 4830, pp. 732–738. Springer, Heidelberg (2007)
11. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall, Upper Saddle River (1999)
12. Reichheld, F.F.: The loyalty effect: the hidden force behind growth, profits, and lasting value. Harvard Business School Press, Boston (2001)
13. Abe, N., Verma, N., Schroko, R., Apte, C.: Cross channel optimized marketing by reinforcement learning. In: Proceedings of the KDD, pp. 767–772 (2004)
14. Sun, P.: Constructing Learning Models from Data: The Dynamic Catalog Mailing Problem. Ph.D thesis, Tsinghua University (2003)
15. Pfeifer, P.E., Carraway, R.L.: Modelling customer relationships as markov chains. Journal of Interactive Marketing 14(2), 43–55 (2000)

Applications of Reinforcement Learning to Structured Prediction

Francis Maes, Ludovic Denoyer, and Patrick Gallinari

LIP6 - University Pierre et Marie Curie
104 avenue du President Kennedy, Paris, France

Abstract. Supervised learning is about learning functions given a set of input and corresponding output examples. A recent trend in this field is to consider structured outputs such as sequences, trees or graphs. When predicting such structured data, learning models have to select solutions within very large discrete spaces. The combinatorial nature of this problem has recently led to learning models integrating a search component.

In this paper, we show that Structured Prediction (SP) can be seen as a sequential decision problem. We introduce SP-MDP: a Markov Decision Process based formulation of Structured Prediction. Learning the optimal policy in SP-MDP is shown to be equivalent as solving the SP problem. This allows us to apply classical Reinforcement Learning (RL) algorithms to SP. We present experiments on two tasks. The first, sequence labeling, has been extensively studied and allows us to compare the RL approach with traditional SP methods. The second, tree transformation, is a challenging SP task with numerous large-scale real-world applications. We show successful results with general RL algorithms on this task on which traditional SP models fail.

1 Introduction

Supervised Learning focuses on learning a function given a set of input-output examples. A lot of models have been proposed mainly for continuous output tasks (*i.e.* regression) or classification tasks. A recent trend is to consider structured outputs such as sequences, trees or graphs. Structured Prediction (SP) is motivated by the amount of applications which are naturally described with such data. Such applications include prediction of protein structure in bio-informatics, image restoration, speech processing, handwriting recognition and several natural language processing tasks such as part-of-speech tagging, named entity extraction, sentence parsing or automatic translation. For example, in the handwriting recognition task, illustrated in figure 1, inputs are sequences of handwritten characters (*e.g.* gray-scale bitmaps) and outputs are sequences of labels identifying recognized characters. Most SP challenges come from the combinatorial nature of the prediction process *i.e.* the number of possible outputs is usually exponential with respect to the size of the input.

Some new task-independent SP models have been proposed during the last years. These models have shown to perform well on simple SP tasks like sequence

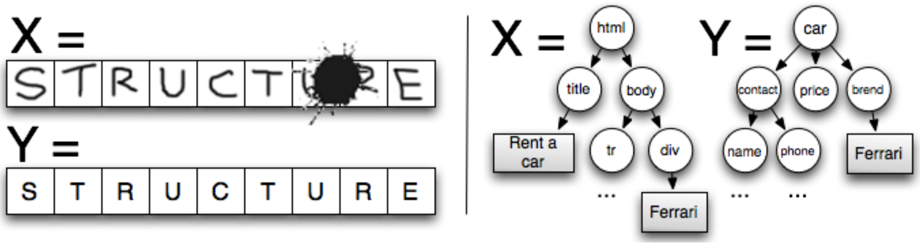


Fig. 1. Two examples of Structured Prediction. Left: sequence labeling, the input **X** is a sequence of handwritten characters and the output **Y** is a sequence of labels identifying the recognized characters. Right: tree transformation, **X** is a an HTML tree from the web and **Y** is an XML tree with additional semantic information.

labeling but they often suffer from scaling problems on more complex tasks. In order to treat SP tasks with larger output spaces, a recent idea is to consider the prediction process as a sequential decision problem. For example, in handwriting recognition, one can build the correct output by taking one decision per character (recognize an 's', recognize an 't', ...). A more complex example is tree transformation, where decisions correspond to node creations, displacements or deletions. In order to construct the output structure, these decisions are performed sequentially. Decisions are interdependent, for example in figure 1 (left), the decision concerning partially hidden characters can be made easier thanks to previous easy character-recognition decisions.

In this paper, we investigate the application of classical reinforcement learning algorithms to SP tasks. Therefore, we introduce a new framework called SP-MDP which is a formulation of SP based on Markov Decision Processes. We show the equivalence between learning the optimal policy in SP-MDP and minimizing the empirical risk of the corresponding SP problem. Thanks to this equivalence, we can apply general reinforcement learning technics to SP problems. Our experiments focus on the two SP tasks illustrated in figure 1. Sequence labeling is probably the simplest and most studied SP task. Our experiments show the competitiveness of general RL algorithms against state-of-the-art SP models on this task. Tree transformation is a challenging SP task. The specific applications considered here imply trees containing more than one thousand nodes, very large description spaces (up to 10^6 dimensions) and an infinite state space. General RL algorithms show successful results on several real-world applications where previous SP methods failed.

The paper is structured as follows. First, we introduce the field of SP and give an overview of existing methods in section 2. We then describe the SP-MDP formulation and show the equivalence between learning a policy in SP-MDP and solving the SP problem in section 3. We demonstrate the interest of the RL approach, with several experiments on sequence labeling and tree transformation tasks in section 4.

2 Structured Prediction

In this section, we define Structured Prediction and give an overview of existing SP methods. We focus on *task-independant* SP models: those which are not restricted to a particular data structure.

2.1 Formalism

The aim of SP is to learn a function which maps inputs $\mathbf{x} \in \mathcal{X}$ to outputs $\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}$, where the outputs are objects which have a structure, such as sequences, trees or graphs. $\mathcal{X}$ is the set of all possible inputs and $\mathcal{Y}_{\mathbf{x}}$ is the set of candidate outputs for a given input $\mathbf{x}$. We denote by $\mathcal{Y} = \cup_{\mathbf{x} \in \mathcal{X}} \mathcal{Y}_{\mathbf{x}}$ the full output space.

For learning, the user supplies a set of examples $D = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i \in [1,n]}$. These examples are supposed to be independently and identically sampled from an unknown distribution $\mathcal{D}_{\mathcal{X} \times \mathcal{Y}}$. In order to evaluate the quality of a prediction, we use a loss function $\Delta(\hat{\mathbf{y}}, \mathbf{y}^*)$ which quantifies how bad it is to predict $\hat{\mathbf{y}}$ instead of $\mathbf{y}^*$. The models introduced in the following are parameterized by a vector $\theta \in \mathcal{R}^d$ and we denote f_θ the prediction function corresponding to parameters θ. Two problems have to be solved in SP:

• **Inference.** Given the parameters θ and an input $\mathbf{x}$, the inference consists in selecting an output $\hat{\mathbf{y}}$ among all candidates $\mathcal{Y}_{\mathbf{x}}$. In most SP problems, the number of candidates is exponential in the size of the input. The predicted output, denoted $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$, should have a low loss value $\Delta(\hat{\mathbf{y}}, \mathbf{y}^*)$.

• **Training.** Given the set of examples D and the loss function Δ, training is the process of searching the best parameters θ that minimize a global loss over the training set. Formally, learning corresponds to minimizing the *expected risk* defined as follows:

$$\theta^* = \operatorname*{argmin}_{\theta \in \mathcal{R}^d} E_{\mathbf{x},\mathbf{y} \sim \mathcal{D}_{\mathcal{X} \times \mathcal{Y}}} \{\Delta(f_\theta(\mathbf{x}), \mathbf{y})\}$$

Since the distribution $\mathcal{D}_{\mathcal{X} \times \mathcal{Y}}$ is unknown, the *expected risk* cannot be computed and the methods usually try to minimize the *empirical risk* defined over the training set:

$$\theta^* = \operatorname*{argmin}_{\theta \in \mathcal{R}^d} \frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}^*) \in D} \Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y}^*)$$

2.2 Compatibility Based Models

One of the first ideas of SP was to generalize existing classification methods to structured outputs. Some new methods have been proposed which are based on a compatibility function $F(\mathbf{x}, \mathbf{y}; \theta)$ that measures *how good is the output y given an input x*. The classification function is thus defined as:

$$f_\theta(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} F(\mathbf{x}, \mathbf{y}; \theta)$$

A common choice is to choose F as a linear function:

$$F(\mathbf{x}, \mathbf{y}; \theta) = < \theta, \phi(\mathbf{x}, \mathbf{y}) >$$

where $< .,. >$ denotes the scalar product and ϕ is an input-output *joint-description* function. Such a function jointly describes an input $\mathbf{x}$ and a corresponding candidate output $\mathbf{y}$ as a feature vector in $\mathcal{R}^d$. Compatibility-based models differ in the meaning which is associated to the compatibility function and in the way to learn the θ parameters.

The Structured Perceptron [1] is a generalization of the classical Perceptron which was first applied to natural language parsing. Learning is performed by simulating inference, and correcting the weight vector each time a wrong output is predicted. No special meaning is given to the compatibility function except the requirement that the correct output should have higher scores than wrong outputs.

Conditional Random Fields (CRFs) [2] use a log-linear probability function to model the conditional probability of an output $\mathbf{y}$ given an input $\mathbf{x}$. CRFs are graphical models where Markov assumptions are used in order to make inference tractable. This allows to express the probability of an output as a product over output sub-structures.

Several methods extending the ideas of Support Vector Machines to SP have been proposed. SVM for Interdependent and Structured Output spaces (SVM-ISO, also known as SVM^{struct}) [3] is a natural generalization of maximum margin classification to structured outputs. Maximum Margin Markov Network (M^3N) [4] is a generalization of probabilistic graphical models which is trained with a discriminant criterion. Both methods try to find parameters θ that lead to large margins: *i.e.* large score differences between correct and wrong outputs. They differ in the way constraints are defined and handled for the optimization.

All compatibility-based models assume that inference can be solved efficiently. This optimization step which is involved in both learning and inference, has mostly been tackled with dynamic programming techniques. This has two major drawbacks. First, dynamic programming requires strong independence assumptions which are often unrealistic for real-world data. For example, in sequence labeling, it is often assumed that a label only interacts with the previous and next labels. Second, even with such independence assumptions, dynamic programming algorithms may have a prohibitive complexity. For example, when predicting trees, the best dynamic programming algorithms have a cubic complexity in the number of leaf nodes. This limits the use of such methods to small trees, *e.g.* less than 50 nodes. This family of methods is unable to deal with large-scale collections and complicated structured output and are limited to simple tasks like sequence labeling for example.

2.3 Incremental Models

Recently, a new idea has emerged from the field of SP. Instead of modeling a compatibility function and then solving a complex optimization problem, it is possible to directly learn the prediction process. Instead of modeling *what a good prediction looks like* the *Incremental models* directly model *how to build the good prediction.* This simple idea thus suggests to integrate learning and searching into a sequential prediction process. This process starts with an empty initial partial output. Each decision correspond to an elementary modification of the output being predicted. States contain *partial outputs* and final states contain *complete outputs.*

Incremental Parsing [5] is one of the first models using the idea of incremental prediction. This model was introduced in the context of natural language parsing, where inputs are sequences of words and outputs are parse trees. Incremental Parsing is built around a Perceptron which assigns ranking scores to partial outputs. The inference is performed greedily by making decisions that maximize the immediate ranking score. Learning is performed by repeating the following process: run the inference procedure until a wrong decision happens, stop inference and make an elementary correction of the Perceptron.

Incremental prediction was popularized by LaSO [6] which is probably the first general Incremental SP model. LaSO relies on a beam-search procedure. The selection of partial outputs in the beam-search is computed by using a Perceptron. LaSO assumes that the whole path leading to the correct output is known for all examples. Learning repeats the following steps until convergence: run the inference procedure until the correct path leaves the beam, make an elementary correction of the Perceptron, re-insert the correct path in the beam and continue.

Searn [7] is another general Incremental SP model developed later. In Searn, the decision maker is modelled by a classifier of any type (*e.g.* Support Vector Machines or Decision Trees). Searn assumes that for each learning example, we know an optimal decision maker. This decision maker knows the best decision to perform for all states of the prediction space. Searn uses an iterative batch-learning approach. At each Searn iteration, a mixture of the optimal decision maker and the learnt decision maker is used to perform inference on all learning examples. For each visited state, one classification example is created. At the end of the iteration, a new classifier is learnt on the basis of these new classification examples. This is repeated until convergence. Searn has shown to be efficient on numerous task and is considered as a general-purpose state-of-the-art SP model.

3 Structured Prediction with Markov Decision Processes

In this section, we introduce a new formulation of SP that is based on Markov Decision Processes. SP-MDPs are MDPs that model the inference process of a structured prediction. The main contribution of this paper is to show that learning the optimal policy in an SP-MDP is equivalent to solving the corresponding SP problem, *i.e.* minimizing the empirical risk.

This original formulation allows to apply general-purpose Reinforcement Learning algorithms to SP tasks. In section 4, we show the competitiveness of such general algorithms against SP state-of-the-art methods. Contrary to LaSO

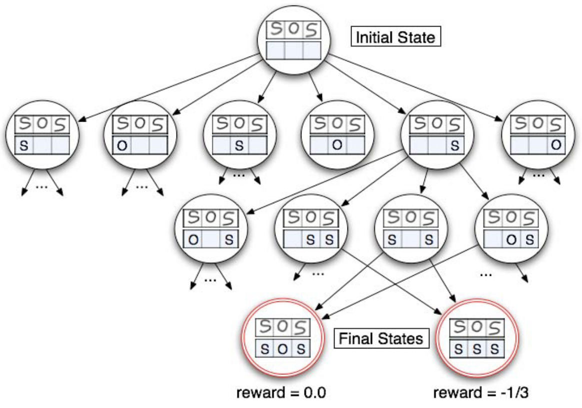


Fig. 2. Handwritten recognition SP-MDP. Circles are states and links are transitions. Each state contains the input and a partial output. Here, the input is a sequence of three *black-and-white* bitmaps representing handwritten digits. Partial outputs are partially recognized sequences of digits. The bottom double circled states are final states containing complete outputs.

and Searn, our approach of SP does not rely on any additional assumption such as the availability of optimal trajectories or the availability of the optimal policy.

3.1 SP Markov Decision Process

We adopt the formalism of deterministic MDPs $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$ where $\mathcal{S}$ is the state-space, $\mathcal{A}$ is the set of possible actions, $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function between states and $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the reward function.

A SP-MDP is a deterministic MDP which models inference for a given SP task. SP-MDPs are illustrated in figure 2 and defined formally below:

• **States.** Each state of a SP-MDP contains both an input $\mathbf{x}$ and a partial output $\bar{\mathbf{y}}$. Let $\bar{\mathcal{Y}}$ be the set of all possible partial outputs. The set of states of a SP-MDP is then $\mathcal{S} = \mathcal{X} \times \bar{\mathcal{Y}}$. There is one initial state per possible input $\mathbf{x}$: $s^{initial}(\mathbf{x}) = (\mathbf{x}, \epsilon)$ where ϵ is the initial empty solution. A set of examples $D = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i \in [1,n]}$ can thus be mapped to a set of corresponding SP-MDP initial states. Final states contain complete outputs which can be returned to the user.

• **Actions.** Actions of SP-MDP concern elementary modifications of the partial output $\bar{\mathbf{y}}$. Those elementary modifications are specific to the SP problem. For example, in handwriting recognition, elementary modifications add a single label prediction to the current partial output. See section 4 for other examples of such elementary modifications. Let $\mathcal{M}(\mathbf{x}, \bar{\mathbf{y}})$ be the set of modifications available given the input $\mathbf{x}$ and the partial output $\bar{\mathbf{y}}$. The set $\mathcal{A}_s \subset \mathcal{A}$ of actions available in state s is defined as follows:

$$\mathcal{A}_{s=(\mathbf{x},\bar{\mathbf{y}})} = \mathcal{M}(\mathbf{x}, \bar{\mathbf{y}})$$

• **Transitions.** SP-MDP transitions are deterministic and replace the current partial output by the transformed partial output. Transitions do not change the current input:

$$T((\mathbf{x}, \bar{\mathbf{y}}), a) = (\mathbf{x}, a(\bar{\mathbf{y}}))$$

where $a(\bar{\mathbf{y}})$ denotes the modified partial output.

• **Rewards.** In SP, the aim is to predict outputs which are as similar as possible to the correct outputs. Formally, we want to minimize the expectation of the SP loss function Δ. The reward function of an SP-MDP, which is discussed in next part, is closely related to Δ:

$$r(s = (\mathbf{x}, \bar{\mathbf{y}}), a) = \begin{cases} -\Delta(a(\bar{\mathbf{y}}), \mathbf{y}^*), & \text{if } a \text{ leads to a final state} \\ 0, & \text{for all other states} \end{cases}$$

Note that the correct output $\mathbf{y}^*$ is required to compute this reward function.

3.2 Learning a SP Policy

In this part, we discuss the reward function of SP-MDPs and show that maximizing the expectation of perceived reward is equivalent to minimizing the empirical risk of SP. Remember that, in order to learn, we have access to a database of examples: inputs with their associated correct outputs. Since the computation of the reward requires the correct outputs, the rewards are only known for states that correspond to training examples. As illustrated by figure 3, we thus distinguish two parts of the state-space: the finite *training* subset and the remaining state-space.

The fact that the reward function is only known in a small subset of the state-space is the main particularity of SP-MDPs, when compared to usual sequential

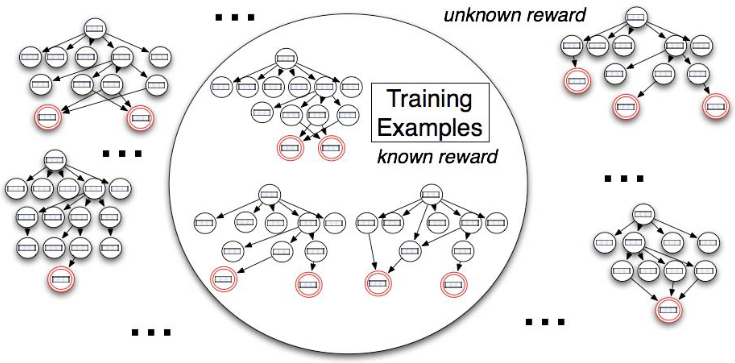


Fig. 3. This figure illustrates a whole SP-MDP. The big circle denotes states that correspond to training SP examples: those where the reward function is known. For all states outside this circle, the reward function is unknown. More than a way to cope with the curse of dimensionality, we use function approximation in order to obtain policies able to generalize on the whole SP-MDP given only a small training subset.

decision problems. The partial observability of the reward function leads us to the following observations. First, policy learning algorithms can only be applied to the training subset of the SP-MDP. Second, we are looking for policies with *generalization capabilities*. The knowledge acquired in the training subset should be applicable to the whole space, in order to make predictions on new inputs.

In order to cope with large state-spaces and with the curse of dimensionality, RL algorithms using function approximation have been developed. We here focus on these algorithms for another reason: function approximation allows to learn the policy on a subset of the MDP and then to *generalize on the whole MDP*.

Approximated RL algorithms learn parameterized policies π_θ where $\theta \in \mathcal{R}^d$ is the vector of parameters. The aim is to find a policy π^*_θ that maximizes the expectation of cumulative reward, given an initial state distribution $\mathcal{D}_0$:

$$\pi^*_\theta = \underset{\theta}{\operatorname{argmax}} \, E_{s_0 \sim \mathcal{D}_0}\{R(\pi_\theta, s_0)\}$$

where the cumulative reward $R(\pi, s_0)$ is the sum of rewards perceived when following π starting from $s_0 : \sum_{t=0}^{T} r(s_t, a_t)$.

We now show that finding the optimal policy in a SP-MDP is equivalent to minimizing the *empirical risk* (see part 2.1) of the SP problem. The key is that the cumulative reward of one episode in SP-MDP is the negative loss of the predicted output:

$$\begin{aligned} R(\pi_\theta, s_0) &= r(s_0, a_0) + \cdots + r(s_{T-1}, a_{T-1}) + r(s_T, a_T) \\ &= 0 + \cdots + 0 - \varDelta(a(\bar{\mathbf{y}}), \mathbf{y}^*) \\ &= -\varDelta(\bar{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y}^*) \end{aligned}$$

Let us consider the distribution $\mathcal{D}^D_0$ which uniformly picks examples from D to build initial states $s^{initial}(\mathbf{x})$. When training a RL algorithm with the $\mathcal{D}^D_0$ initial states distribution, we are looking for the optimal policy:

$$\begin{aligned} \pi^*_\theta &= \underset{\theta \in \mathcal{R}^d}{\operatorname{argmax}} \, E_{s^{initial}(\mathbf{x}) \sim \mathcal{D}^D_0}\{R(\pi_\theta, s_0)\} \\ &= \underset{\theta \in \mathcal{R}^d}{\operatorname{argmin}} \, E_{s^{initial}(\mathbf{x}) \sim \mathcal{D}^D_0}\{\varDelta(\bar{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y}^*)\} \\ &= \underset{\theta \in \mathcal{R}^d}{\operatorname{argmin}} \, \frac{1}{N} \sum_{(\mathbf{x}, \mathbf{y}^*) \in D} \varDelta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y}^*) \end{aligned}$$

The *optimal policy* with the initial state distribution $\mathcal{D}^D_0$ is thus *the policy which minimizes the empirical risk*. This equivalence enables the use of general RL algorithms with function approximation like Sarsa, QLearning, Monte Carlo Control or TD(λ) [8].

3.3 Representations

In order to use function approximation, we need an action-description function $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}^d$. This function describes state-action pairs with a set of d

feature-values. A feature can describe any joint aspect of the action and the current state. In the experiments described in section 4, we use large sparse feature spaces. Typically, the dimensionality d ranges from 10^3 to 10^6. In order to build features, we use *feature-templates* and *feature-generators*.

Consider for example the handwriting recognition task, where input characters are black-and-white bitmaps and actions concern single label predictions. In such a task, we want to have one feature per pixel and per label. We then introduce for example the following feature-template:

$$f_{x,y,l}(s,a) = \begin{cases} 1, & \text{if (the pixel } (x,y) \text{ is black)} \wedge \text{(the predicted label is } l\text{)} \\ 0, & \text{otherwise} \end{cases}$$

In order to describe a state-action pair, we use a feature-generator function which enumerates all the features that follow the feature-template and which are non-null. One example of such automatically generated feature is given below:

$$f_{3,7,'S'}(s,a) = \begin{cases} 1, & \text{if (the pixel (3,7) is black)} \wedge (a = \text{'S'}) \\ 0, & \text{otherwise} \end{cases}$$

If we consider 10×10 black-and-white pixels and 26 possible labels, there are $d = 2600$ distinct features. However, for a given state-action pair, all the features corresponding to other labels than the selected one and those corresponding to white pixels are null. This leads to a sparse representation of less than 100 non-null features. In practice, sparsity allows for an efficient implementation where only the non-null features are taken into account. In our experiments, we use one to ten manually defined task-dependent feature-templates. This is enough to automatically generate up to millions of distinct features. Contrary to compatibility-based models which make decomposability assumptions on the description function (see part 2.2), the SP-MDP approach allows to include any long-term dependency in the description function.

Sparse high-dimensional representations were introduced in the context of natural language processing [9] and have good practical properties: they are fast to compute and allow the use of simple, but powerful, linear learning machines.

4 Experiments

In this section, we describe several experiments performed with the SP-MDP approach on two SP tasks: sequence labeling and tree transformation. We introduce two main results in this section. Firstly, with the classical SP task of sequence labeling, we show the competitiveness of RL-based algorithms in comparison to state-of-art SP methods. Secondly, with the tree transformation task, we give a successful example of the capability of general RL algorithms to treat very large-scale SP tasks. Indeed, tree transformation deals with large trees (thousand nodes), complex transformations (structure and text processing, node creations, deletions and displacements) and very high dimensional learning (more than one million dimensions). Up to our knowledge, SP-MDP is the only existing model able to handle this task yet. Traditional SP methods fail on this task, either due to their restrictive assumptions or due to their excessive complexity.

4.1 Sequence Labeling

Sequence labeling amounts at predicting a label sequence $\mathbf{y} = (y_1, \ldots, y_T)$ given an observation sequence $\mathbf{x} = (x_1, \ldots, x_T)$. Each label y_t corresponds to the observation x_t and belongs to the set of possible labels $\mathcal{L}$. Sequence labeling has various applications such as handwriting recognition, information extraction, named entity extractions or sentence chunking.

We now detail the initial outputs, the actions and the loss function which define the sequence labeling SP-MDP. In order to represent partial outputs, we introduce a particular label $\perp$ which denotes variables y_t which have not been labeled yet.

• **Initial Outputs.** The initial partial outputs in sequence labeling are sequences where no labels have been decided yet. An initial output ϵ can thus be written $\epsilon = (\perp, \perp, \ldots, \perp)$.

• **Actions.** Actions in sequence labeling correspond to single label prediction. We have compared two sets of actions: *Left-to-right* labeling and *Order-free* labeling. In the former, the first label y_1 is decided at the first step, the second label y_2 at the second step and so forth. At each step, there are $card(\mathcal{L})$ possible actions which correspond to all possible labels for the current element. In order-free labeling, any unlabeled element can be labeled at any time (as in figure 2). This allows the system to first perform easy decisions (*e.g.* recognize easy letters) in order to gain more context for harder decisions (*e.g.* recognize a partially hidden letters). More details about this approach are provided in [10].

• **Loss function.** In order to evaluate the quality of a particular labeling, we use the Hamming Loss Δ. This loss function simply returns the number of wrong predicted elements: $\Delta(\hat{\mathbf{y}}, \mathbf{y}^*) = card(\{i \in [1,T], \hat{y}_i \neq y_i^*\})$.

• **Datasets.** We performed our experiments on three classical sequence labeling datasets:

- **Spanish Named Entity Recognition (NER).** This dataset, introduced in the CoNLL 2002 shared task[1], is made of spanish sentences where the aim is to find persons, locations and organisms names (there are 9 distinct labels in $\mathcal{L}$). We used two train/test splits NER-large (8,324 training sentences) and NER-small (300 training sentences), as in [7] and [3]. Features include the words, the prefixes and suffixes of the words in a window of +/- 2 words.
- **Chunk.** This dataset put forward by [11] and introduced in the CoNLL-2000 shared task[2] is composed of sections of the Wall Street Journal corpus. The aim is to split sentences into non-overlapping nominal groups. There are

[1] http://www.cnts.ua.ac.be/conll2002/ner/
[2] http://www.cnts.ua.ac.be/conll2000/chunking/

three labels (BIO encoding): Begin of a new group, Inside a group, Outside a group. Input features include the words, prefixes, suffixes and *part-of-speech* of surrounding word.

– **HandWritten.** This corpus was created for handwritting recognition and was introduced by [12]. It includes 6,600 sequences of handwritten characters corresponding to 6,600 words collected from 150 subjects. Each word is composed of letters, which are 8×16 pixels images, rasterized into a binary representation. As in [7], we used two variants of the set: HandWritten-small (10% words for training) and HandWritten-large (90% words for training). Letters are described using one feature per pixel.

Table 1. Top: percentage of correctly predicted labels on the testing set per dataset and model. Bottom: training times of each model with a traditional desktop machine. On some datasets, denoted by –, SVM^{struct} required too much memory to be applicable.

	Left To Right			Order Free		Non-incremental	
	Sarsa	RankingRL	Searn	Sarsa	RankingRL	CRF	SVM^{struct}
NER-small	91.90	**93.67**	**93.8**	91.28	93.35	91.86	93.45
NER-large	96.31	**96.94**	96.3	96.32	96.75	**96.96**	-
HandWritten-small	68.41	74.01	64.1	70.63	73.57	66.86	**76.94**
HandWritten-large	80.33	83.80	73.5	79.59	**84.08**	75.45	-
Chunk	96.08	96.22	95.0	96.17	**96.54**	**96.71**	-
NER-large	$\approx$ 35min	$\approx$ 25min	$\approx$ 6h	$\approx$ 11h	$\approx$ 8h	$\approx$ 8h	$>$ 3 days
HandWritten-large	$\approx$ 15min	$\approx$ 12min	$\approx$ 3h	$\approx$ 6h	$\approx$ 4h	$\approx$ 2h	$>$ 3 days

• **Models.** Experiments have been performed with the *left-to-right* and *order-free* models combined with two algorithms: an approximated Sarsa(0) algorithm and the RankingRL algorithm proposed in [10]. Briefly, instead of learning an action-value which is the expectation of future discounted rewards, this algorithm learns to rank actions directly: only the order of the learnt *action-scores* matters. In all cases, we use ϵ-greedy sampling, where ϵ decreases exponentially with the number of learning iterations. The discount and learning rate parameters have been tuned manually. We compare the RL-based methods with two non-incremental state-of-the-art sequence labeling models: Conditional Random Fields[3] and SVM^{struct} [4] (see part 2.2). We also compare with an existing Incremental SP method: Searn[5] (see part 2.3).

• **Results.** The results of our experiments are given in table 1. On all datasets, the RL based methods are competitive with state-of-the-art sequence labeling methods which demonstrates clearly the interest of our approach. *order-free* and *left-to-right* models give similar results on these experiments; see [10] for a discussion on these variants. Furthermore, for comparable performance, we have significantly lower training times than those of the non-Incremental models. It

[3] FlexCRFs implementation: http://flexcrfs.sourceforge.net.

[4] SVM^{struct} implementation: http://svmlight.joachims.org/svm_struct.html

[5] Searn implementation: http://searn.hal3.name

should also be noted, that inference in incremental model is several orders faster, since it simply consists in greedily executing the learnt policy (instead of solving a global optimization problem). These low training and inference times, allow us to consider much harder tasks than the simple sequence labeling, such as the tree transformation task described below.

4.2 Tree Transformation

We consider here a *tree transformation* task where both the inputs and outputs are ordered labeled trees. The applications we present deal with semi-structured textual documents. They consist in converting weakly structured (flat text or HTML) documents into highly structured XML documents. The main challenges of this tree transformation task come from the size of the documents, the complexity of the transformations and the huge number of training examples.

In order to perform tree transformation, we represent the input trees as a sequence of *in-context* leaves: $\mathbf{x} = (x_1, \ldots, x_T)$. Each leaf x_i is a feature vector describing the leaf textual content and the context of the leaf (*e.g.* parent labels and sibling labels).

• **Initial Outputs.** The initial outputs are composed of a single root node.

• **Actions.** The key idea of our SP-MDP is that one leaf x_t is processed per time step t. Processing a leaf means adding its textual content somewhere into the partial output tree. Either the textual content can be added into an existing node, or new nodes can be created to store the textual content. An action consists in two steps: first it selects an internal node of the partial output tree; then it creates new nodes from this location until a leaf where the textual content is added. This leads to a very large action space which is determined by a simple preprocessing of the examples dataset (see [13] and [14]). We furthermore consider a `SKIP` action which allows to ignore the current textual content.

• **Loss function.** In order to evaluate the quality of an output, we used three tree-similarity functions. $F_{content}$ measures the proportion of correctly labeled leaves. F_{path} measures the proportion of correctly recovered paths. A path is a sequence of labels from the root node until a leaf node. $F_{structure}$ measures the proportion of correctly recovered subtrees. The latter is very strict and decreases quickly with only a few errors. For a single labeling error in a leaf, $F_{structure}$ typically equals to $\approx 80\%$. See [13] for more details. The loss function used in learning is the negative $F_{structure}$ score between the predicted and the correct output.

• **Datasets.** We used four large-scale real-world datasets and one small dataset:

– **INEX IEEE [15].** The INEX IEEE corpus is composed of 12017 scientific articles in XML format, coming from 18 different journals. The documents are given in two versions: a flat segmented version and the XML version. The tree transformation task aims at recovering the XML structure using only the text segments as input.

– **Mixed Movies [15].** The second corpus is made of more than 13000 movie descriptions available in three versions: two mixed different XHTML versions and one XML version. This corresponds to a scenario where two different websites have to be mapped onto a predefined mediated schema. The transformation includes node suppression and some node displacements.

- **Wikipedia [16].** This corpus is composed of 12,000 wikipedia pages. For each page, we have one XML representation dedicated to wiki-text and the HTML code. The aim is to use the low-level information available in the HTML version, to predict an high-level representation.
- **RealEstate.**[6] This corpus, proposed by Anhai Doan is made of 2,367 data-oriented XML documents. The documents are expressed in two different XML formats. The aim is to learn the transformation from the format to the other.
- **Shakespeare**[7] **[17].** This corpus is composed of 60 Shakespearean scenes. These scenes are small trees, with an average length of 85 leaf nodes and 20 internal nodes over 7 distinct tags.

• **Models.** Each corpus is split into two parts: 50% for training and 50% for testing. Our model is the approximated-Sarsa(0) algorithm applied to the SP-MDP described above. We only have one baseline on two datasets because most existing SP methods do not scale with our large datasets. The baseline itself does not scale on our three large datasets. The model PCFG+ME [17] can be seen as a compatibility based model for the tree transformation problem. It models the probability of outputs by using probabilistic context free grammars (PCFG) and maximum-entropy (ME) classifiers. Inference is then performed with a dynamic-programming algorithm which has a cubic complexity in the number of input leaves. LaSO cannot be applied because we do not have access to correct paths for learning examples (see part 2.3). Searn cannot be applied because we do not have access to an optimal decision maker. Indeed, given a partial output tree and the target output tree, the optimal decisions are those which minimize the tree-edit distance between both trees. The best algorithms that compute tree-edit distances have at least a cubic complexity in the number of nodes. Due to the large size of our trees, these computations are not tractable so that we cannot compute the optimal decisions.

• **Results.** The results of our experiments are given in table 2. All $F_{content}$ scores are greater than 75 % while the more difficult $F_{structure}$ is still greater than $\approx$ 60 %. These scores are encouraging when considering the intrinsic difficulty of the tree transformation tasks. With INEX IEEE for example, the only hints for predicting one label among more than 100 labels come from the textual content of the input document (length, case, first and last words, ...).

On the small datasets, the scores of Sarsa are slightly lower than those of the PCFG+ME baseline. This may due to the fact that PCFG+ME performs a global optimization using dynamic programming whereas Sarsa performs a greedy inference of the output tree. Greedy inference slightly degrades performance but it brings speed (most of document are inferred in less than one second)

[6] http://www.cs.wisc.edu/ anhai/

[7] http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

Table 2. This table summarizes our experiments. From left to right: the dataset, dataset statistics (the number of documents, the average number of nodes in correct output documents and the number of distinct labels in output documents), the model and the three average similarity scores between predicted and correct outputs on the test set ($F_{content}$, F_{path} and $F_{structure}$).

Dataset	Size	AvNodes	NumLabels	Model	$F_{content}$	F_{path}	$F_{structure}$
INEX IEEE Flat → XML	12,017	≈ 700	139	Sarsa	75.8 %	74.4 %	67.5 %
Mixed Movies HTML → XML	13,038	≈ 100	40	Sarsa	79.2 %	77.8 %	64.5 %
Wikipedia HTML → XML	12,000	≈ 200	256	Sarsa	80.2 %	74.3 %	65.6 %
RealEstate XML→ XML	2,367	≈ 34	37	Sarsa	99.9 %	99.9 %	99.9 %
				PCFG+ME	99.9 %	7 %	49.8 %
Shakespeare Flat → XML	60	≈ 105	7	Sarsa	94.4 %	93.2 %	87.5 %
				PCFG+ME	98.7 %	97%	94.7 %

and scalability. Finally, an interesting result is that with a general reinforcement learning, we are able to solve a problem where previous SP methods failed.

5 Conclusion

In this paper, we introduced the structured prediction problem and gave an overview of some well-known methods for solving it. We introduced the SP-MDP formulation of structured prediction, which allows to apply general reinforcement learning algorithms to sequence, tree or graph prediction problems. Our experiments show that these general RL technics are often competitive against specialized SP algorithms. Furthermore, our approach requires less assumptions than the previous SP approaches, which allows us to deal with a complex and large scale tree transformation task.

From the point of view of reinforcement learning, our problems are original on a number of aspects: the reward is only known for a subset of the space, the MDPs are discrete and very-large and we put a special emphasis on the generalization capabilities of the learnt policies. Given this unusual setting, we believe that the explicit bridge between SP and reinforcement learning may lead both domains to cross-fertilize and mutually reinforce each other.

References

1. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: EMNLP (2002)
2. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: ICML 2001 (2001)
3. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: ICML 2004 (2004)

4. Taskar, B., Guestrin, C., Koller, D.: Max-margin markov networks. In: NIPS (2003)
5. Collins, M., Roark, B.: Incremental parsing with the perceptron algorithm. In: ACL 2004, Barcelona, Spain, pp. 111–118 (July, 2004)
6. Daumé III, H., Marcu, D.: Learning as search optimization: Approximate large margin methods for structured prediction. In: ICML, Bonn, Germany (2005)
7. Daumé III, H., Langford, J., Marcu, D.: Search-based structured prediction (2006)
8. Sutton, R., Barto, A.: Reinforcement learning: an introduction. MIT Press, Cambridge (1998)
9. Berger, A., Della Pietra, S., Della Pietra, V.: A maximum entropy approach to natural language processing. In: Computational Linguistics (1996)
10. Maes, F., Denoyer, L., Gallinari, P.: Sequence labelling with reinforcement learning and ranking algorithms. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS, vol. 4701, pp. 648–657. Springer, Heidelberg (2007)
11. Ramshaw, L., Marcus, M.: Text chunking using transformation-based learning. In: Yarovsky, D., Church, K. (eds.) Proceedings of the Third Workshop on Very Large Corpora, Somerset, New Jersey, ACL, pp. 82–94 (1995)
12. Kassel, R.H.: A comparison of approaches to on-line handwritten character recognition. Ph.D thesis, Cambridge, MA, USA (1995)
13. Maes, F., Denoyer, L., Gallinari, P.: Xml structure mapping application to the pascal/inex 2006 xml document mining track. In: INEX, Dagstuhl, Germany (2007)
14. Maes, F., Denoyer, L., Gallinari, P.: Apprentissage de conversions de documents semi-structures a partir d'exemples. In: CORIA, Tregastel, France (2008)
15. Denoyer, L., Gallinari, P.: Report on the xml mining track at inex 2005 and inex 2006. SIGIR Forum, 79–90 (2007)
16. Denoyer, L., Gallinari, P.: The wikipedia xml corpus. SIGIR Forum (2006)
17. Chidlovskii, B., Fuselier, J.: A probabilistic learning method for xml annotation of documents. In: IJCAI (2005)

Policy Learning – A Unified Perspective with Applications in Robotics

Jan Peters[1,2], Jens Kober[1], and Duy Nguyen-Tuong[1]

[1] Max-Planck Institute for Biological Cybernetics,
Spemannstr. 32, 72074 Tübingen
[2] University of Southern California,
Los Angeles, CA 90089, USA
{jrpeters,kober,duy}@tuebingen.mpg.de
http://kyb.mpg.de/∼jrpeters

Abstract. Policy Learning approaches are among the best suited methods for high-dimensional, continuous control systems such as anthropomorphic robot arms and humanoid robots. In this paper, we show two contributions: firstly, we show a unified perspective which allows us to derive several policy learning algorithms from a common point of view, i.e, policy gradient algorithms, natural-gradient algorithms and EM-like policy learning. Secondly, we present several applications to both robot motor primitive learning as well as to robot control in task space. Results both from simulation and several different real robots are shown.

1 Introduction

In order to ever leave the well-structured environments of factory floors and research labs, future robots will require the ability to aquire novel behaviors, motor skills and control policies as well as to improve existing ones. Reinforcement learning is probably the most general framework in which such robot learning problems can be phrased. However, most of the methods proposed in the reinforcement learning community to date are not applicable to robotics as they do not scale beyond robots with more than one to three degrees of freedom. Policy learning methods are a notable exception to this statement. Starting with the pioneering work of Gullapali, Franklin and Benbrahim [4, 8] in the early 1990s, these methods have been applied to a variety of robot learning problems ranging from simple control tasks (e.g., balancing a ball-on a beam [3], and pole-balancing [11]) to complex learning tasks involving many degrees of freedom such as learning of complex motor skills [8, 15, 20] and locomotion [6, 7, 10, 13, 16, 22, 24].

In this paper, we expand previous work on policy learning towards the direction of a unified framework for policy learning. For doing so, we discuss upper and lower bounds on policy improvements. From the lower bound, we derive a cost function which allows us to derive policy gradient approaches, natural policy gradient approaches as well as EM-like policy learning methods. Furthermore, we show several applications in the context of robot skill learning. These applications include both learning task-space control with reinforcement learning as well as motor primitive learning. Results of both real robots and simulation are being shown.

2 Policy Learning Approaches

As outlined before, we need two different styles of policy learning algorithms, i.e., methods for long-term reward optimization and methods for immediate improvement. We can unify this goal by stating a cost function

$$J(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, r(\boldsymbol{\tau})\, d\boldsymbol{\tau}, \tag{1}$$

where $\boldsymbol{\tau}$ denotes a path, e.g., $\boldsymbol{\tau} = [\mathbf{x}_{1:n}, \mathbf{u}_{1:n}]$ with states $\mathbf{x}_{1:n}$ and actions $\mathbf{u}_{1:n}$, $r(\boldsymbol{\tau})$ denotes the reward along the path, e.g., $r(\boldsymbol{\tau}) = \sum_{t=1}^{n} \gamma^t r_t$ and $p_{\boldsymbol{\theta}}(d\boldsymbol{\tau})$ denotes the path probability density $p_{\boldsymbol{\theta}}(d\boldsymbol{\tau}) = p(\mathbf{x}_1) \prod_{t=1}^{n-1} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)\, \pi(\mathbf{u}_t|\mathbf{x}_t; \boldsymbol{\theta})$ with a first-state distribution $p(\mathbf{x}_1)$, a state transition $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ and a policy $\pi(\mathbf{u}_t|\mathbf{x}_t; \boldsymbol{\theta})$. Note, that $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, r(\boldsymbol{\tau})$ is an improper distribution, i.e., does not integrate to 1. The policy $\pi(\mathbf{u}_t|\mathbf{x}_t; \boldsymbol{\theta})$ is the function which we intend to learn by optimizing its parameters $\boldsymbol{\theta} \in \mathbb{R}^N$. Many policy learning algorithms have started optimize this cost function, including policy gradient methods [1], actor-critic methods [14, 23], the Natural Actor-Critic [19, 20, 21] and Reward-Weighted Regression [18]. In the remainder of this section, we will sketch a unified approach to policy optimization which allows the derivation of all of the methods above from the variation of a single cost function. This section might appear rather abstract in comparison to the rest of the paper; however, it contains major novelties as it allows a coherent treatment of many previous and future approaches.

2.1 Bounds for Policy Updates

In this section, we will look at two problems in policy learning, i.e., an upper bound and a lower bound on policy improvements. The upper bound outlines why a greedy operator is not a useful solution while the lower bound will be used to derive useful policy updates.

Upper Bound on Policy Improvements. In the stochastic programming community, it is well-known that the greedy approach to policy optimization suffers from the major drawback that it can return only a biassed solution. This drawback can be formalized straighforwardly by showing that if we optimize $J(\boldsymbol{\theta})$ and approximate it by samples, e.g., by $\hat{J}_S(\boldsymbol{\theta}) = \sum_{s=1}^{S} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}_s)\, r(\boldsymbol{\tau}_s) \approx J(\boldsymbol{\theta})$, we obtain the fundamental relationship

$$E\{\max_{\boldsymbol{\theta}} \hat{J}_S(\boldsymbol{\theta})\} \geq \max_{\boldsymbol{\theta}} E\{\hat{J}_S(\boldsymbol{\theta})\}, \tag{2}$$

which can be shown straightforwardly by first realizing the that the maximum is always larger than any member of a sample. Thus, a subsequent expectation will not change this fact nor the subsequent optimization of the lower bound. Thus, a policy which is optimized by doing a greedy step in parameter space is guaranteed to be biased in the presence of errors with a bias of $b_S(\boldsymbol{\theta}) = E\{\max_{\boldsymbol{\theta}} \hat{J}_S(\boldsymbol{\theta})\} - \max_{\boldsymbol{\theta}} E\{\hat{J}_S(\boldsymbol{\theta})\} \geq 0$. However, we can also show that the bias decreases over the number of samples, i.e., $b_S(\boldsymbol{\theta}) \geq b_{S+1}(\boldsymbol{\theta})$, and converges to zero for infinite samples, i.e., $\lim_{S\to\infty} b_S(\boldsymbol{\theta}) = 0$ [17]. This optimization bias illustrates the deficiencies of the greedy operator: for finite data any policy update is problematic and can result into unstable learning processes with oscillations, divergence, etc as frequently observed in the reinforcement learning community [1, 2].

Lower Bound on Policy Improvements. In other branches of machine learning, the focus has been on lower bounds, e.g., in Expectation-Maximization (EM) algorithms. The reasons for this preference apply in policy learning: if the lower bound also becomes an equality for the sampling policy, we can guarantee that the policy will be improved. Surprisingly, the lower bounds in supervised learning can be transferred with ease. For doing so, we look at the scenario (suggested in [5]) that we have a policy $\boldsymbol{\theta}'$ and intend to match the path distribution generated by this policy to the success weighted path distribution, then we intend to minimize the distance between both distributions, i.e., $D\left(p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)r\left(\boldsymbol{\tau}\right)\right)$. Surprisingly, this results into a lower bound using Jensen's inequality and the convexity of the logarithm function. This results into

$$\log J(\boldsymbol{\theta}') = \log \int \frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)}{p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)} p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right) d\boldsymbol{\tau}, \tag{3}$$

$$\geq \int p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right) \log \frac{p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)}{p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)} d\boldsymbol{\tau} \propto -D\left(p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right)\right), \tag{4}$$

where $D\left(p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)\right) = \int p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) \log(p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)/p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)) d\boldsymbol{\tau}$ is the Kullback-Leibler divergence, i.e., a distance measure for probability distributions. With other words, we have the lower bound $J(\boldsymbol{\theta}') \geq \exp\left(D\left(p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right)\right)\right)$, and we can minimize

$$J_{\text{KL}} = D\left(p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right)\right) = \int p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right) \log \frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right) r\left(\boldsymbol{\tau}\right)}{p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)} d\boldsymbol{\tau} \tag{5}$$

without the problems which have troubled the reinforcement learning community when optimizing the upper bound as we are guaranteed to improve the policy. However, in many cases, we might intend to punish divergence from the previous solution. In this case, we intend to additionally control the distance which we move away from our previous policy, e.g., minimize the term $J_{+} = -D\left(p_{\boldsymbol{\theta}}\left(\boldsymbol{\tau}\right)||p_{\boldsymbol{\theta}'}\left(\boldsymbol{\tau}\right)\right)$. We can combine these into a joint cost function

$$J_{\text{KL}+} = J_{\text{KL}} + \lambda J_{+}, \tag{6}$$

where $\lambda \in \mathbb{R}^{+}$ is a positive punishment factor with $0 \leq \lambda \leq J(\boldsymbol{\theta})$. Note that the exchange of the arguments is due to the fact that the Kullback-Leibler divergence is unsymmetric. This second term will play an important rule as both baselines and natural policy gradients are a directly result of it. The proper determination of λ is non-trivial and depends on the method. E.g., in policy gradients, this becomes the baseline.

2.2 Resulting Approaches for Policy Learning

We now proceed into deriving three different methods for lower bound optimization, i.e., policy gradients, the natural actor-critic and reward-weighted regression. All three of these can be derived from this one perspective.

Policy Gradients Approaches. It has recently been recognized that policy gradient methods [1, 2] do not suffer from the drawbacks of the greedy operator and, thus, had a large revival in recent years. We can derive policy gradient approaches straightforwardly from this formulation using the steepest descent of the first order taylor extension

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha(\boldsymbol{\nabla} J_{\mathrm{KL}} - \lambda \boldsymbol{\nabla} J_{+}) \tag{7}$$

$$= \boldsymbol{\theta} + \alpha \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\left(r(\boldsymbol{\tau}) - \lambda\right) \boldsymbol{\nabla} \log p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})\, d\boldsymbol{\tau}, \tag{8}$$

where α is a learning rate. This is only true as for the first derivative $\boldsymbol{\nabla} D(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})||p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}))$ $= \boldsymbol{\nabla} D\left(p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right)$. The punishment factor from before simply becomes the baseline of the policy gradient estimator. As $\boldsymbol{\nabla} \log p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}) = \sum_{t=1}^{n-1} \boldsymbol{\nabla} \log \pi(\mathbf{u}_t|\mathbf{x}_t;\boldsymbol{\theta})$, we obtain the straightforward gradient estimator also known as REINFORCE, policy gradient theorem or GPOMDP, for an overview see [1]. The punishment term only constrains the variance of the policy gradient estimate and vanishes as $\boldsymbol{\nabla} J_{\mathrm{KL}+} = \boldsymbol{\nabla} J_{\mathrm{KL}}$ for infinite data. However, this policy update can be shown to be rather slow [9, 19, 20, 21].

Natural Policy Gradient Approaches. Suprisingly, the speed update can be improved significantly if we punish higher order terms of J_+, e.g., the second term of the taylor expansion yields

$$\boldsymbol{\theta}' = \operatorname{argmax}_{\boldsymbol{\theta}'}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T(\boldsymbol{\nabla} J_{\mathrm{KL}} - \lambda \boldsymbol{\nabla} J_{+}) - \frac{1}{2}\lambda(\boldsymbol{\theta}' - \boldsymbol{\theta})^T \boldsymbol{\nabla}^2 J_{+}(\boldsymbol{\theta}' - \boldsymbol{\theta}) \tag{9}$$

$$= \lambda\left(\boldsymbol{\nabla}^2 J_{+}\right)^{-1}(\boldsymbol{\nabla} J_{\mathrm{KL}} - \lambda \boldsymbol{\nabla} J_{+}) = \lambda \mathbf{F}^{-1} g_1, \tag{10}$$

where $\mathbf{F} = \boldsymbol{\nabla}^2 D\left(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})||p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})\right) = \boldsymbol{\nabla}^2 D\left(p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right) = \boldsymbol{\nabla}^2 J_{+}$ is also known as the Fisher information matrix and the resulting policy update $\mathbf{g}_2$ is known as the Natural Policy Gradient. Surprisingly, the second order term has not yet been expanded and no Natural second-order gradient approaches are known. Thus, this could potentially be a great topic for future research.

EM-Policy Learning. In a very special case, we can solve for the optimal policy parameters, e.g, for policy which are linear in the log-derivatives such as

$$\boldsymbol{\nabla} \log \pi(\mathbf{u}_t|\mathbf{x}_t;\boldsymbol{\theta}) = \mathbf{A}\left(\mathbf{x}_t, \mathbf{u}_t\right)\boldsymbol{\theta} + \mathbf{b}\left(\mathbf{x}_t, \mathbf{u}_t\right), \tag{11}$$

it is straightforward to derive an EM algorithm such as

$$\boldsymbol{\theta}' = \alpha^{-1}\beta, \tag{12}$$

$$\boldsymbol{\alpha} = \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\left(r(\boldsymbol{\tau}) - \lambda\right)\sum_{t=1}^{n} \mathbf{A}\left(\mathbf{x}_t, \mathbf{u}_t\right) d\boldsymbol{\tau}, \tag{13}$$

$$\boldsymbol{\beta} = \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\left(r(\boldsymbol{\tau}) - \lambda\right)\sum_{t=1}^{n} b\left(\mathbf{x}_t, \mathbf{u}_t\right) d\boldsymbol{\tau}. \tag{14}$$

This type of algorithms can result into very fast policy updates if applicable. It does not require a learning rate and is guaranteed to converge to at least a locally optimal solution.

2.3 Sketch of the Resulting Algorithms

Thus, we have developed two different classes of algorithms, i.e., the Natural Actor-Critic and the Reward-Weighted Regression.

Natural Actor-Critic. The Natural Actor-Critic algorithms [19, 20] instantiations of the natural policy gradient previously described with a large or infinite horizon n. They are considered the fastest policy gradient methods to date and "the current method of choice" [1]. They rely on the insight that we need to maximize the reward while keeping the loss of experience constant, i.e., we need to measure the distance between our current path distribution and the new path distribution created by the policy. This distance can be measured by the Kullback-Leibler divergence and approximated using the Fisher information metric resulting in a natural policy gradient approach. This natural policy gradient has a connection to the recently introduced compatible function approximation, which allows to obtain the Natural Actor-Critic. Interestingly, earlier Actor-Critic approaches can be derived from this new approach. In application to motor primitive learning, we can demonstrate that the Natural Actor-Critic outperforms both finite-difference gradients as well as 'vanilla' policy gradient methods with optimal baselines.

Reward-Weighted Regression. In contrast to Natural Actor-Critic algorithms, the Reward-Weighted Regression algorithm [18] focuses on immediate reward improvement, i.e., $n = 1$, and employs an adaptation of the expectation maximization (EM) policy learning algorithm for reinforcement learning as previously described instead of a gradient based approach. The key difference here is that when using immediate rewards, we can learn from our actions directly, i.e., use them as training examples similar to a supervised learning problem with a higher priority for samples with a higher reward. Thus, this problem is a reward-weighted regression problem, i.e., it has a well-defined solution which can be obtained using established regression techniques. While we have given a more intuitive explanation of this algorithm, it corresponds to a properly derived maximization-maximization (MM) algorithm which maximizes a lower bound on the immediate reward similar to an EM algorithm. Our applications show that it scales to high dimensional domains and learns a good policy without any imitation of a human teacher.

Policy Learning by Weighting Exploration with Rewards. A recent development is the policy learning by weighting exploration with rewards or PoWER method [12]. In this case, we attempt to extend the previous work of the reward-weighted regression from the immediate reward case to longer horizons. When using the reward-weighted regression, we suffer from a multitude of artificial local plateaus and will not converge to the optimal solution. However, the insight that state-dependent exploration rates result into this algorithm. Again, an EM algorithm is obtained and turns out to be highly efficient in the context of learning Kendama [12].

3 Robot Application

The general setup presented in this paper can be applied in robotics using analytical models as well as the presented learning algorithms. The applications presented in this paper include motor primitive learning and operational space control.

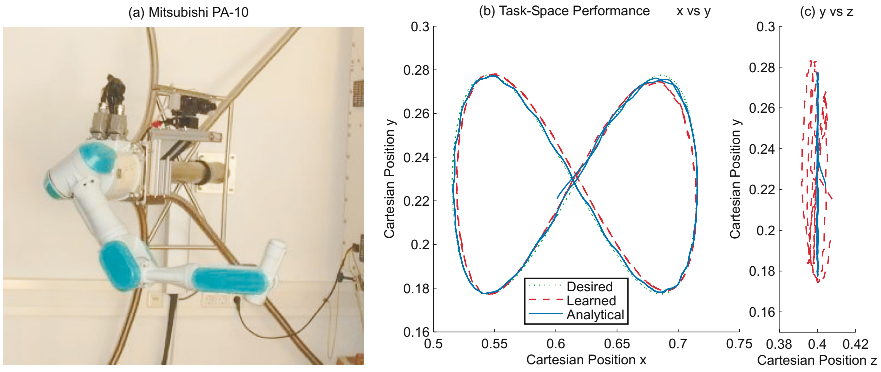


Fig. 1. (a) Mitsubishi PA-10 robot arm with seven degrees of freedom used in the experiments in this paper. (b) This figure illustrates the task performance of both the analytical and the learned resolved velocity control laws. Here, the green dotted line shows the desired trajectory which the robot should follow, the red dashed line is the performance of the real-time learning control law while the blue solid line shows the performance of the resolved velocity control law. Note, that while the online learning solution is as good as the analytical solution, it still yields comparable performance without any pre-training of the local control laws before the online learning (Nevertheless, the predictors were pre-trained).

3.1 Learning Operational Space Control

Operational space control is one of the most general frameworks for obtaining task-level control laws in robotics. In this paper, we present a learning framework for operational space control which is a result of a reformulation of operational space control as a general point-wise optimal control framework and our insights into immediate reward reinforcement learning. While the general learning of operational space controllers with redundant degrees of freedom is non-convex and thus global supervised learning techniques cannot be applied straightforwardly, we can gain two insights, i.e., that the problem is locally convex and that our point-wise cost function allows us to ensure global consistency among the local solutions. We show that this can yield the analytically determined optimal solution for simulated three degrees of freedom arms where we can sample the state-space sufficiently. Similarly, we can show the framework works well for simulations of the both three and seven degrees of freedom robot arms as presented in Figure 1.

3.2 Motor Primitive Improvement by Reinforcement Learning

The main application of our long-term improvement framework is the optimization of motor primitives. Here, we follow essentially the previously outlined idea of acquiring an initial solution by supervised learning and then using reinforcement learning for motor primitive improvement. For this, we demonstrate both comparisons of motor primitive learning with different policy gradient methods, i.e., finite difference methods, ‘vanilla’ policy gradient methods and the Natural Actor-Critic, as well as an application of the most successful method, the Natural Actor-Critic to T-Ball learning on a physical, anthropomorphic SARCOS Master Arm, see Figure 2.

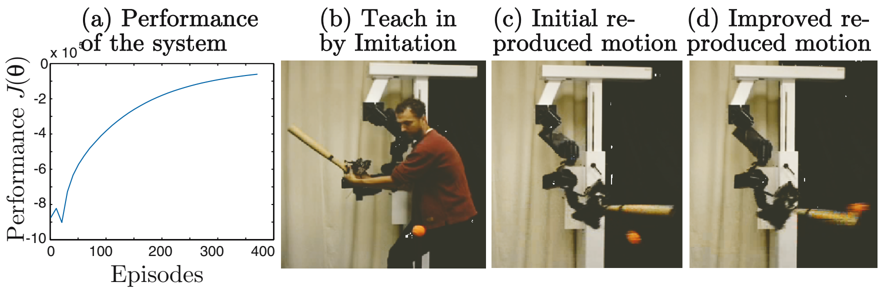


Fig. 2. This figure shows (a) the performance of a baseball swing task when using the motor primitives for learning. In (b), the learning system is initialized by imitation learning, in (c) it is initially failing at reproducing the motor behavior, and (d) after several hundred episodes exhibiting a nicely learned batting.

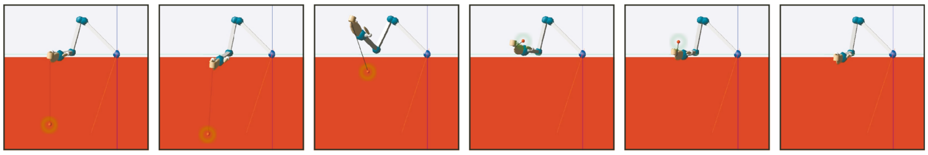


Fig. 3. This figure illustrates the successfully learned motion of the Kendama trial. For achieving this motion, motor primitives with external feedback had to be learned. Only an imitation from a human trial recorded in a VICON setup and, subsequently, reinforcement learning allowed to learn this motion reliably.

Another example for applying policy learning to the motor primitive frame is the children's game Kendama [12]. Here, we have managed to learn a good policy again from a human demonstration which fails to bring the ball into the cup. Subsequently, we have learned how to improve our policy with the PoWER method [12] and have managed to learn a good motor primitive-based control policy. The results are shown in Figure 3.

4 Conclusion

In conclusion, in this paper, we have presented a general framework for learning motor skills which is based on a thorough, analytically understanding of robot task representation and execution. We have introduced a general framework for policy learning which allows the derivation of a variety of novel reinforcement learning methods including the Natural Actor-Critic and the Reward-Weighted Regression algorithm. We demonstrate the efficiency of these reinforcement learning methods in the application of learning to hit a baseball with an anthropomorphic robot arm on a physical SARCOS master arm using the Natural Actor-Critic, and in simulation for the learning of operational space with reward-weighted regression.

References

1. Aberdeen, D.: POMDPs and policy gradients. In: Proceedings of the Machine Learning Summer School (MLSS), Canberra, Australia (2006)
2. Aberdeen, D.A.: Policy-Gradient Algorithms for Partially Observable Markov Decision Processes. Ph.D thesis, Australian National Unversity (2003)
3. Benbrahim, H., Doleac, J., Franklin, J., Selfridge, O.: Real-time learning: A ball on a beam. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Baltimore, MD (1992)
4. Benbrahim, H., Franklin, J.: Biped dynamic walking using reinforcement learning. Robotics and Autonomous Systems 22, 283–302 (1997)
5. Dayan, P., Hinton, G.E.: Using expectation-maximization for reinforcement learning. Neural Computation 9(2), 271–278 (1997)
6. Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., Cheng, G.: Learning cpg sensory feedback with policy gradient for biped locomotion for a full-body humanoid. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Pittsburgh, PA (2005)
7. Geng, T., Porr, B., Wörgötter, F.: Fast biped walking with a reflexive neuronal controller and real-time online learning. In: Int. Journal of Robotics Res. (submitted, 2005)
8. Gullapalli, V., Franklin, J., Benbrahim, H.: Aquiring robot skills via reinforcement learning. IEEE Control Systems Journal, Special Issue on Robotics: Capturing Natural Motion 4(1), 13–24 (1994)
9. Kakade, S.A.: Natural policy gradient. In: Advances in Neural Information Processing Systems, Vancouver, CA, vol. 14 (2002)
10. Kimura, H., Kobayashi, S.: Reinforcement learning for locomotion of a two-linked robot arm. In: Birk, A., Demiris, J. (eds.) EWLR 1997. LNCS, vol. 1545, pp. 144–153. Springer, Heidelberg (1998)
11. Kimura, H., Kobayashi, S.: Reinforcement learning for continuous action using stochastic gradient ascent. In: Proceedings of the International Conference on Intelligent Autonomous Systems (IAS), Madison, Wisconsin, vol. 5, pp. 288–295 (1998)
12. Kober, J., Peters, J.: Reinforcement learning of perceptual coupling for motor primitives. In: The European Workshop on Reinforcement Learning, EWRL (submitted, 2008)
13. Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA (May 2004)
14. Konda, V., Tsitsiklis, J.: Actor-critic algorithms. Advances in Neural Information Processing Systems 12 (2000)
15. Mitsunaga, N., Smith, C., Kanda, T., Ishiguro, H., Hagita, N.: Robot behavior adaptation for human-robot interaction based on policy gradient reinforcement learning. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Edmonton, Canada, pp. 1594–1601 (2005)
16. Mori, T., Nakamura, Y., Sato, M.-a., Ishii, S.: Reinforcement learning for cpg-driven biped robot. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), San Jose, CA, pp. 623–630 (2004)
17. Peters, J.: The bias of the greedy update. Technical report, University of Southern California (2007)
18. Peters, J., Schaal, S.: Learning operational space control. In: Proceedings of Robotics: Science and Systems (RSS), Philadelphia, PA (2006)
19. Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS), Karlsruhe, Germany (September 2003)

20. Peters, J., Vijayakumar, S., Schaal, S.: Natural actor-critic. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS, vol. 3720, pp. 280–291. Springer, Heidelberg (2005)
21. Richter, S., Aberdeen, D., Yu, J.: Natural actor-critic for road traffic optimisation. In: Schoelkopf, B., Platt, J.C., Hofmann, T. (eds.) Advances in Neural Information Processing Systems, vol. 19. MIT Press, Cambridge (2007)
22. Sato, M.-a., Nakamura, Y., Ishii, S.: Reinforcement learning for biped locomotion. In: Dorronsoro, J.R. (ed.) ICANN 2002. LNCS, vol. 2415, pp. 777–782. Springer, Heidelberg (2002)
23. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Solla, S.A., Leen, T.K., Mueller, K.-R. (eds.) Advances in Neural Information Processing Systems (NIPS), Denver, CO. MIT Press, Cambridge (2000)
24. Tedrake, R., Zhang, T.W., Seung, H.S.: Learning to walk in 20 minutes. In: Proceedings of the Yale Workshop on Adaptive and Learning Systems. Yale University, New Haven (2005)

Probabilistic Inference for Fast Learning in Control

Carl Edward Rasmussen[1,2] and Marc Peter Deisenroth[1,3]

[1] Department of Engineering, University of Cambridge, UK
[2] Max Planck Institute for Biological Cybernetics, Tübingen, Germany
[3] Faculty of Informatics, Universität Karlsruhe (TH), Germany

Abstract. We provide a novel framework for very fast model-based reinforcement learning in continuous state and action spaces. The framework requires probabilistic models that explicitly characterize their levels of confidence. Within this framework, we use flexible, non-parametric models to describe the world based on previously collected experience. We demonstrate learning on the cart-pole problem in a setting where we provide very limited prior knowledge about the task. Learning progresses rapidly, and a good policy is found after only a hand-full of iterations.

1 Introduction

Learning from experience is a key ingredient in the behavior of intelligent beings and holds great potential for artificial systems. Algorithms for learning from experience are studied in the areas of reinforcement learning (RL) and adaptive control. A central issue for such algorithms is the *speed of learning*, that is, the number of trials necessary to learn a task. Many learning algorithms require a huge number of trials to succeed, whereas biological systems often learn quickly.

There are broadly two types of approaches to speed up learning of artificial systems. One approach is to constrain the task in various ways to simplify learning. The issue with this approach is that it is highly problem dependent and relies on an a priori understanding of the characteristics of the task. Alternatively, one can speed up learning by extracting more useful information from available experience. This effect can be achieved by carefully modeling the observations. In a practical application, one would typically combine these two approaches. In this paper, we are concerned solely with the second approach: How can we learn as fast as possible, given only very limited prior understanding of a task? Thus, we are not looking for an engineering solution to a particular problem, but rather we elicit a general algorithm for effective learning. The approach is general. For purposes of illustration, we will apply it to the well-known cart-pole problem.

1.1 Related Work

Experience from real interactions can be used for two purposes. It can be used either to update the current model of the world (indirect RL) or it can be used to improve the value function and/or policy directly (direct RL), or combinations of the two. With a learned model, it is possible to get *simulated* experience to perform a planning update resulting in a new (model-specific) policy and/or value function. Real experience is used to build the model and not to directly optimize the policy itself. This idea is described by Sutton's Dyna architecture introduced in [1].

Direct and indirect methods have different strengths and weaknesses. On the one hand, model-free algorithms do not rely on a (possibly incorrect) model. On the other hand, they require many interactions with the real system to find a solution to the considered RL problem. In real-world problems, hundreds of thousands or millions of interactions with the real system are often infeasible due to time, physical, and monetary constraints. In contrast to model-free methods, indirect (model-based) approaches can make more efficient use of limited experience. However, they may suffer if the model employed is not a sufficiently good approximation to the real world. This problem was already recognized by Atkeson and Santamaría [2] and Atkeson and Schaal [3]. To overcome the problem of policies for inaccurate models, Abbeel et al. add a heuristic bias term when updating the model after gathering real experience, [4]. In [5], Poupart et al. learn a probabilistic model for a finite state POMDP problem to incorporate observations into prior knowledge. However, a principled and rigorous way of building models that consistently quantify knowledge and uncertainty does not exist in the RL literature to our best knowledge. In our approach, we use flexible non-parametric probabilistic models to reap the benefit of the indirect approach while minimizing any problems of model bias.

Traditionally, solving even relatively simple tasks from scratch have been considered "daunting", [6], in the absence of strong task-specific prior assumptions. In the context of robotics, one popular solution employs prior knowledge provided by a human "teacher" to restrict the solution space [3,4,6,7,8]. The teacher shows the robot what a possible solution looks like by demonstrating it. Building a (local) model using data from this demonstration might also alleviate model errors along a good trajectory.

In contrast to previous work, we propose a fast RL algorithm which is able to learn a good policy *without* problem-specific prior knowledge. Similar to [9] and [10], we consider model-based policy iteration using probabilistic dynamics models. However, in [9,10], actions are considered as parameters to be optimized in any state rather than modeling the policy as a function of the state. In contrast to [9] and [10], we propose learning the policy explicitly.

2 Fast Reinforcement Learning Framework

For fast reinforcement learning, we propose an adaptive probabilistic world model learned on previous experience. Due to limited experience, a probabilistic model is required to appropriately model what is known and what not. The model is used in a planning step to determine a good (model-optimized) policy.

Algorithm 1. Fast reinforcement learning

```
1: initial exploration                                        ▷ interaction phase
2: loop
3:     collect observations
4:     update probabilistic dynamics model
5:     optimize policy through simulation                     ▷ planning phase
6:     apply (model-optimized) policy to real system          ▷ interaction phase
7: end loop
```

A high-level description of the general framework is given in Algorithm 1. We distinguish between the phase of real interaction and simulation in the planning phase. The algorithm is initialized by an arbitrary policy and some notion about the current state of the world. In the interaction phase, we take this information and apply a single action to the real world according to our current policy. We observe a new state and employ the policy again. Subsequently, the probabilistic world model is updated using new and historical experience collected during the interaction phases. During the planning phase, we take a state distribution and *simulate* the world with the corresponding distribution over actions. The probabilistic world model determines a distribution over successor states. The controller computes the corresponding distribution over costs and the system is being simulated by applying the policy to the entire state distribution.

During learning, the policy will be optimized in the planning phase based on the evidence so far. However, it may be that because of limited experience the simulations do not correspond to the real world. When this situation occurs and we apply the model-optimized policy to the real system, the model will discover the discrepancy between the model's predictions and the world. This new insight will be incorporated into the subsequent model update.

Crucially, in order for the internal simulations to reflect the real world as accurately as possible, the dynamics models must faithfully represent their fidelities of how accurate they are. For example, if a state is visited on a simulated trajectory about which not much knowledge has been acquired, the model must be able to quantify this uncertainty, and not simply assume that its best guess is close to the truth. A probabilistic model quantifies knowledge and can be considered as a model that captures all plausible models in a distribution over models. The use of probabilistic models for the dynamics allows us to keep track of the uncertainties in the simulations used for planning. Typically, in early stages of learning, the uncertainty in the states will grow with increasing prediction horizon. When applying a good real-world policy on the other hand, we expect the uncertainty to collapse because the system is being controlled. With increasing experience, the probabilistic model will tend to a deterministic one.

Modeling a dynamic system is generally fairly easy for short time horizons, but gets progressively more challenging as the horizon increases. Therefore, we are forced to learn the dynamics model on relatively short time scales. However, good control strategies often require the consideration of long-term effects of immediate actions. To bridge this gap, we need to cascade many short-term predictions to assess long-term behavior during the planning phase.

Our approach explicitly requires a probabilistic world model in the planning step although the framework resembles Sutton's Dyna architecture introduced in [1]. However, we believe that utilizing a deterministic model is inconsistent when uncertainties are involved.

Cost Function. Let us revisit Algorithm 1. In the planning stage, the policy is optimized to minimize the expected long-term cost starting from an initial state distribution. The expected immediate cost is a function of the state. Traditionally, the squared error cost function has been extensively used. However, because of its convexity and unboundedness, the cumulative cost will be highly dependent on the worst state along a trajectory. Initially, when the dynamics model is uncertain, the uncertainty may grow rapidly with the time horizon, so that the expected cost will be highly sensitive to details of a distribution which essentially encodes that the model has "lost track of the state". To avoid the extreme dependence on these essentially arbitrary details, we use instead an immediate cost function which is locally quadratic, but which saturates for large deviations from the desired goal.

Stochastic Simulation. Although the policy is a deterministic function of the state, the simulation of trajectories involves distributions over trajectories. The states are uncertain since the learned dynamics model is probabilistic. Intuitively speaking, as the probabilistic model is a distribution over all plausible models, a deterministic state is being mapped through all plausible transition functions resulting in a distribution over successors states. In the simulation phase, the distribution over states thus implies a distribution over actions, even when the policy is deterministic. When the policy is applied to the real system where the current state has just been measured, a single action is applied deterministically.

3 Implementation

In the following, we describe how to implement the general ideas from the previous section in discrete-time setting with continuous states and actions.

The probabilistic short-term dynamics models are implemented using flexible non-parametric models based on Gaussian processes (GPs). State uncertainty is explicitly propagated forward to obtain a probabilistic representation of long-term behavior. These computations can be done approximately in closed form, and Markov chain Monte Carlo is not necessary. Conditioned on the probabilistic dynamics model, the policy is optimized using policy iteration. Since the implicit internal simulation of the system can be done analytically, we have a computationally efficient method to perform a policy evaluation step. Moreover, we can compute the gradient of the expected long-term cost with respect to the policy parameters analytically, which allows the use of standard optimization methods, such as conjugate gradients.

3.1 Dynamics Model

We propose learning the dynamics using Gaussian process models. A GP is a distribution over functions and utilized for state-of-the-art Bayesian non-parametric regression. Regarding a function as an infinitely long vector, all necessary computations can be broken down to manipulating standard Gaussian distributions. Thus, GP regression combines both flexible non-parametric modeling and tractable Bayesian inference. For further details, please refer to [11].

The GP dynamics model takes as input a representation of the current state and action. As output the GP computes a representation of the distribution over consecutive states. In particular, for a D-dimensional state, we utilize D separate GPs, one for each state dimension, explicitly incorporating correlations between state variables. The dynamics models are learned using the observed state trajectories by the standard algorithm (evidence maximization), see [11].

In the planning stage, the predicted state trajectories are uncertain. We therefore need to be able to predict outputs when the inputs are uncertain. To carry out the necessary computations, we use results from Girard et al., [12], and Kuss [9]. Generally, a Gaussian state followed by a nonlinear dynamics results in a non-Gaussian successor state. We follow the above references and compute exactly the two first moments of the resulting distribution, that is, a Gaussian approximation. By iteration, we can thus compute the distribution over trajectories in closed form.

Throughout all computations, we explicitly take the uncertainty about the dynamics into account by averaging over all plausible dynamics models given the current set of observations from the real world. The variances of the predicted successor states take into account both the uncertainty in the current state *and* the possibly imprecise model of the actual dynamics.

3.2 Policy Model

The controller implements a nonlinear deterministic policy. While any function approximator can be used for this purpose, we apply (the mean of) a non-parametric Gaussian process policy model. This policy GP is parameterized by a (pseudo-) training set, consisting of pairs of states (training inputs) and corresponding actions (training targets). By modifying this training set, we can control the policy being implemented. This idea is related to the sparse Gaussian process approximation using inducing inputs introduced by Snelson and Ghahramani in [13]. In contrast to [13], we do not average over the training targets, but optimize them as well. Note that since the policy GP predicts deterministically, the uncertainty about the underlying policy is zero.

In this paper, we consider deterministic policies only: For a deterministic input, the policy always returns the same control. However, due to the probabilistic dynamics model there will be uncertainties about states resulting in distributions over actions as described in Section 3.1. As illustrated in Figure 1, during the planning phase, we cascade short-term dynamics models to predict what is going to happen in the long term. When interacting with the real system, we assume

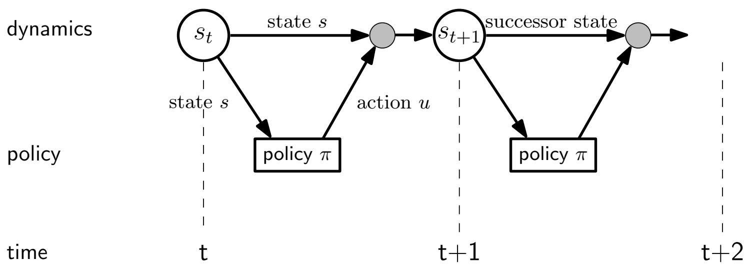


Fig. 1. Interplay of dynamics GP and policy to propagate uncertainty over time. Consider a state distribution $p(\mathbf{s}_t)$ at time step t. The policy takes this distribution as input and returns a distribution over actions. The dynamics model takes a fully joint Gaussian of states and actions (shaded nodes in the figure) as input distribution and determines mean and covariance of the successor state distribution $p(\mathbf{s}_{t+1})$ analytically as detailed in [12]. The distribution $p(\mathbf{s}_{t+1})$ is approximated by a Gaussian with the corresponding mean and covariance.

that the state is measured deterministically. Then, the controller applies the (unique) corresponding control signal. In other words, we apply only a single action deterministically when interacting with the real system, but we consider a distribution over actions in simulation.

3.3 Policy Iteration

Determination of an optimal policy through policy iteration is a natural choice within the proposed framework. In the following, we will show that the policy can be evaluated analytically. Moreover, we provide a framework where the gradient of the expected cumulative cost with respect to the policy parameters can be determined analytically.

Policy Evaluation. Assume a given policy π and a given probabilistic dynamics model. To evaluate the quality of the policy starting from a particular state distribution $p(\mathbf{s}_0)$, the expected cost of the trajectory $\boldsymbol{\tau} := (\mathbf{s}_0, \pi(\mathbf{s}_0), \ldots, \mathbf{s}_T)$ has to be determined. Assuming time-additive losses and a Markovian structure of the problem, the expected undiscounted finite-horizon cost

$$V^{\pi}(\boldsymbol{\tau}) := \mathrm{E}_{\boldsymbol{\tau}}\Big[\sum_{t=0}^{T} \ell(\mathbf{s}_t)\big|\pi, p(\mathbf{s}_0)\Big] = \sum_{t=0}^{T} \mathrm{E}[\ell(\mathbf{s}_t)|\pi, p(\mathbf{s}_0)] \tag{1}$$

has to be evaluated, where the expectation is taken with respect to the probability distribution over trajectories $\boldsymbol{\tau}$. As described in Section 3.1, a Gaussian approximation of the predictive distributions of future states can be determined analytically. To evaluate equation (1), it remains to compute $\mathrm{E}[\ell(\mathbf{s}_t)|\pi, p(\mathbf{s}_0)]$ for an immediate cost function ℓ. If we restrict ℓ for example to contain combinations involving trigonometric functions, exponentials, and powers, this integral is analytically tractable and the gradient of equation (1) with respect to the policy parameters can be computed analytically.

Policy Optimization. To optimize the policy, we minimize the expected long-term cost (1) with respect to the policy parameters using a gradient-based method. Two different kinds of parameters are involved in the Gaussian process controller. Firstly, the hyper-parameters of the kernel and secondly, the pseudo training set of the controller itself. All these parameters are collected inside the parameter vector $\boldsymbol{\theta}$. They are treated as free variables to be optimized to find an optimal strategy. In contrast to standard parameter optimization for GPs by maximizing the marginal likelihood, the objective function in our RL setup is the expected long-term cost given by equation (1).

We employ an efficient conjugate gradients minimizer, which requires the partial derivatives of the objective function with respect to the policy parameters. These derivatives can be computed analytically by repeated application of the chain rule applied to the parameters of the *distributions* governing the states over time. The gradient of V^π, equation (1), with respect to the policy parameters is given by

$$\frac{\mathrm{d}V^\pi(\boldsymbol{\tau})}{\mathrm{d}\boldsymbol{\theta}} = \sum_{t=0}^{T} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\,,$$

where the subscript $\boldsymbol{\theta}$ stresses the dependency of π on the parameter set $\boldsymbol{\theta}$. The total derivative with respect to the policy parameters is denoted by $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}$. As we know the approximate (Gaussian) state distribution $p(\mathbf{s}_t)$, we just have to compute

$$\left(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\mu}_t} \mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\right) \frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} + \left(\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\Sigma}_t} \mathrm{E}[\ell(\mathbf{s}_t)|\pi_{\boldsymbol{\theta}}, p(\mathbf{s}_0)]\right) \frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}}$$

for $t = 0, \ldots, T$, where $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ are mean and covariance of $p(\mathbf{s}_t)$, respectively. Exploiting the Markov property, required computations boil down to

$$\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t}{\partial\boldsymbol{\theta}}\,, \tag{2}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\mu}_{t-1}}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\Sigma}_{t-1}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t}{\partial\boldsymbol{\theta}}\,, \tag{3}$$

where $\boldsymbol{\theta}$ contains all policy parameters and $\frac{\partial}{\partial\boldsymbol{\theta}}$ denotes the partial derivative with respect to the parameter vector $\boldsymbol{\theta}$. Note that the moments of the state distribution $p(\mathbf{s}_t)$ is functionally dependent on the parameter vector $\boldsymbol{\theta}$ and the moments $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$ of the state distribution at time $t-1$. In principle, these computations are straightforward although the details are somewhat lengthy.

4 Experiments

For demonstration purposes, we apply our approach to learning a controller for the underactuated cart-pole problem. The cart-pole task is depicted in Figure 2. As in Doya's paper [14], the pendulum has to be swung up and balanced.

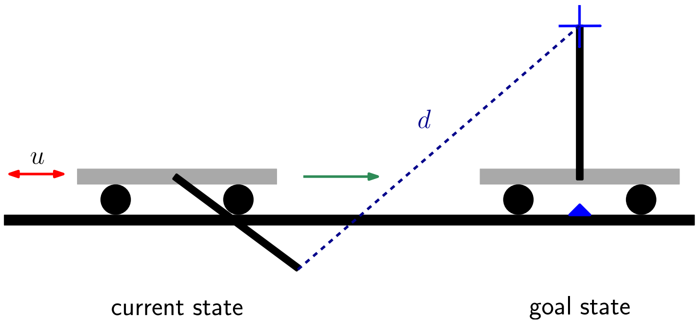


Fig. 2. Cart-pole problem. The pendulum has to be swung up and balanced at the cross by just pushing the cart to left and right. The Euclidean distance between the tip of the pendulum and the goal state is shown by the dashed line.

However, instead of just balancing the pendulum, we additionally require the pendulum to be balanced in a specific location given by the cross in Figure 2. Note that the solution of the task implies that the cart stops at the triangle. One point which makes the problem non-trivial is that to solve it, sometimes actions have to be taken which temporarily move the pendulum further away from the target. Thus, greedily optimizing a short-term cost will lead to a policy that fails to achieve the task. In control theory, the solution of the task is usually based on an intricate understanding of the system dynamics, which we do not assume in this paper. Our objective is to learn a good policy without a prior understanding of the system.

4.1 Cost Function

In general, we are interested in understanding the generic principles, which allow good solutions to be learned automatically. Hence, the only built-in assumption is that the state variables evolve smoothly over time. The state $\mathbf{s}$ of the system consists of cart position x, cart velocity $\dot{x}$, angle[1] of the pole φ, and angular velocity of the pole $\dot{\varphi}$.

The only feedback the controller gets about the quality of its applied action is the squared distance

$$d(\mathbf{s})^2 = x^2 + 2xl\sin(\varphi) + 2l^2 + 2l^2\cos(\varphi)$$

between the tip of the pendulum and its desired position, measured every 200 ms. The distance d is denoted by the dashed line in Figure 2. Note, that in contrast to common implementations, d only depends on the position variables x, $\sin(\varphi)$, and $\cos(\varphi)$. In particular, it does not depend on the velocity variables. We choose the immediate cost

[1] Since the angle is periodic, we actually encode it in the input to the dynamics GP and policy GP as the two variables $\sin(\varphi)$ and $\cos(\varphi)$, to avoid any discontinuity in the representation.

$$\ell(\mathbf{s}) = 1 - \exp(-\tfrac{1}{2}d(\mathbf{s})^2/c^2) \in [0,1]\,, \tag{4}$$

where $c = 0.25\,\text{m}$ is a constant giving the distance at which the cost function switches between locally quadratic and saturation. The cost is zero at the goal state, increases with distance, but saturates at unity.

We generally think this cost is a better cost function than the standard quadratic since for very uncertain states we will get a cost approaching unity reflecting that the state is simply "not good", whereas the quadratic cost would depend crucially on the exact value of the error bar and highly penalize the most extreme state variable.

Our choice of cost function is supposed to be a naïve, intuitive cost which does not rely on an intricate understanding of the physical system. For example, with a little more hindsight we may have chosen to also penalize the velocity variables departure from zero. However, we consider this as a part of the challenge of the learning problem. Our choice of the cost function reflects ignorance about the dynamics as would be the case if we were using our algorithm to solve a complex novel task. Elaborate tuning of the cost function (including also control penalty) has indeed been used extensively in the literature to simplify problems. Here, we derive our cost simply from purely geometric consideration. The only additional information is the constant $c = 0.25\,\text{m}$, giving a rough idea of what it means to be "close" to a solution (note for comparison that the pendulum length is $0.6\,\text{m}$).

4.2 Experimental Setup

The dynamics of the cart-pole system follow the ODEs

$$\begin{aligned}
\ddot{x} &= \frac{\left(\frac{u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2 \sin\varphi}{I + m(\frac{l}{2})^2} + m^2(\frac{l}{2})^2 g \sin\varphi\cos\varphi\right)}{(M+m)(I + m(\frac{l}{2})^2) - m^2(\frac{l}{2})^2(\cos\varphi)^2}\,,\\
\ddot{\varphi} &= \frac{m\frac{l}{2}\cos\varphi(u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2\sin\varphi) + (M+m)gm\frac{l}{2}\sin\varphi}{m^2(\frac{l}{2})^2(\cos\varphi)^2 - (M+m)(I + m(\frac{l}{2})^2)}\,,
\end{aligned}$$

where $M = 0.5\,\text{kg}$ is the mass of the cart, $m = 0.5\,\text{kg}$ the mass of the pole, $b = 0.1\,\text{N s/m}$ the friction between cart and ground, $l = 0.6\,\text{m}$ the length of the pole, $I = 0.06\,\text{kg m}^2$ the moment of inertia around the tip of the pole, and $g = 9.82\,\text{m/s}^2$ the gravitational constant.

The control $u \in [-10, 10]\,\text{N}$ is a horizontal force pushing the cart to left or right. To guarantee that the controller learns and implements only forces in the admissible range $[-u_{\max}, u_{\max}] = [-10, 10]\,\text{N}$, the probability distribution of the control signal is squashed through the sine function, such that $p(u) = p(u_{\max}\sin(\pi(\mathbf{s})))$.

For the dynamics model, we use four separate GP models, one for each state variable. The policy is implemented by a GP, which is parameterized by a pseudo training set of pairs of states and actions. Throughout the experiments, we use a pseudo training set with 50 elements, and accordingly the policy contains approximately 300 free parameters.

The eigen-frequency of this system is of the order of 1 Hz, and we use a short-term prediction time of 0.2 seconds. Note that this is a much slower sampling time than is typically used in conventional controllers for this problem. It is adequate here, as it is easy to capture the dynamics at this time scale. We optimize the objective function (1) over 5 s, that is, 25 time steps. In our setting, the initial state distribution $p(\mathbf{s}_0)$ is Gaussian with zero mean and covariance matrix $\mathbf{\Sigma} = 10^{-4}\mathbf{I}$. The goal state is $\mathbf{s}_{\text{goal}} = [0, 0, \pi, 0]^T$. In other words, the initial state is that the cart is in the right position, but the pendulum is hanging down (instead of being balanced in the upright position). The task for the learning algorithm is thus to explore the state space and to find and to exploit a strategy which will achieve the swing-up and balancing. Note that this task is not achievable by a linear controller.

We implement policy iteration within the fast RL framework given by Algorithm 1 as follows. Initially, we assume fully unknown transition dynamics. To build a first dynamics model, we have to gather some experience. We observe two short (five seconds) trajectories of the system by applying forces randomly starting from an initial state since we do not have prior knowledge about a good policy. We initialize 50 pseudo training inputs of the controller to be the states along the random trajectories. The corresponding pseudo training targets are initialized randomly distributed around zero. In the next step, we build a probabilistic Gaussian process model of the transition dynamics using the observed data. Utilizing this model, we simulate the dynamics for five seconds and optimize the policy parameters using conjugate gradients.[2] Now, an optimized policy for the current dynamics model has been determined, and we are ready to apply the policy to the real system again. The application of the model-optimized policy is presumably not optimal when applied to the real system and, therefore, might lead the real cart-pole system to unexpected states. However, the applied policy is better than just applying random forces, such that states closer to the goal state are visited. We collect these new observations and update the dynamics model by incorporating all experience from previous interactions with the real system.

Alternating, the policy is optimized based on the dynamics model, and the model itself is updated based on collected data when applying the policy to the real system. With each iteration the probabilistic model describes the dynamics better and with more confidence.

4.3 Evaluations

In Figure 3, the predicted immediate costs and the costs incurred when applying the optimized policy to the real system are plotted over a horizon of 5 s. The system is started in the state where the pendulum is hanging down.

In the top left plot, we see that for the first roughly 3 seconds the system does not enter states with a cost significantly different from unity. At about 3 seconds the model predicts a decline in cost, but simultaneously a rapid increase in the

[2] Note that the derivatives (2) and (3) can be computed analytically exactly.

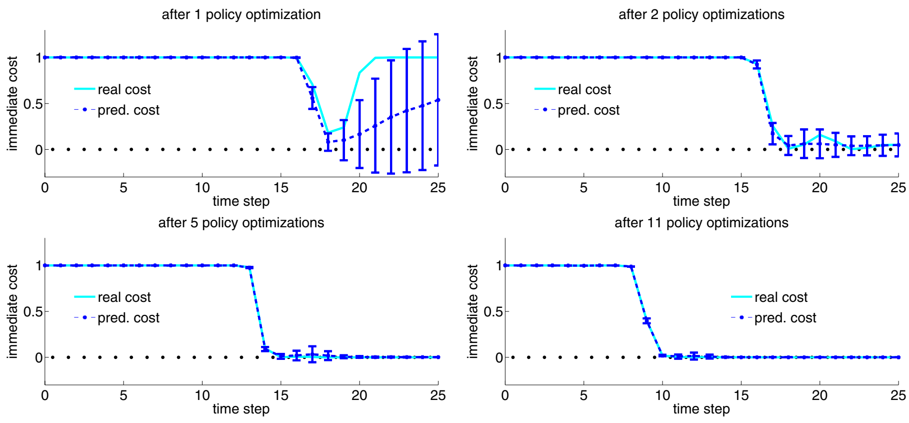


Fig. 3. Predicted costs and real costs after 1, 2, 5, and 11 policy optimizations (from top left to bottom right). The x-axis is the number of 200 ms time steps, the y-axis is the immediate costs. The black dots denote the minimum possible immediate cost when the pendulum is exactly in the goal state. The solid graphs show the real costs, the dashed graphs show the predicted immediate cost distribution. The error bars are twice the standard deviation.

error bars is seen. This reflects the poor dynamics model around the goal state as we have never seen any data in this region. Applying the found policy, we see (in the solid line) that indeed the cost does decrease after about 3 seconds. Furthermore, the actual trajectory lies roughly in agreement with the model's assessment of its own accuracy. In the top left hand plot, the model now has 5 seconds more experience, also including states close to the goal state. The model now predicts much smaller error bars, and that the small cost can be maintained until the end of the simulation. The actual trajectory is in agreement with this. Iterating the learning procedure for more steps results in even smaller error bars and in a quicker swing-up action. The determined policy is not necessarily an optimal one, but it is doing a fairly good job, and the system has found it automatically in less than 60 seconds experience. See `http://mlg.eng.cam.ac.uk/carl/ewrl08` for demos of the learning system. Doya solved a similar problem requiring about 20,000 seconds (approximately 5.5 hours) of interactions with the real system to perform the swing-up reliably, [14].

Note that in the above experiments, we only test whether the system is able to solve the task from a single specific start state. Generally, we would seek solutions applicable to larger regions of the state space. Our algorithm can naturally handle this in two ways: a wider *distribution* of start state can explicitly be specified, or alternatively, multiple paths with different start states (or distributions) can be simulated. Both of these approaches work successfully, although we do not report the results here due to lack of space.

5 Discussion

We have demonstrated that our model-based Bayesian RL algorithm is able to learn a policy for the cart-pole problem from scratch in a handful of iterations. The algorithm carefully models uncertainties about the underlying latent dynamics and takes them seriously into account during planning. In contrast to Doya's results reported in [14], our algorithm is very fast and can solve the swing-up task based on a minute or less interactions with the real system.

It is interesting to speculate what aspects of the model are key to its success. Although we have not yet investigated this fully, we did assess whether it is possible to solve the problem using a deterministic model for the dynamics. We repeated the experiment, but changed the predictions to have zero variance (which corresponds to a deterministic dynamics model). In this case, the algorithm failed completely to solve the task. In particular, for the deterministic model, the policy optimization problem becomes very difficult. This is presumably caused by actions having effects over long horizons, even when the dynamics models are so poor that very little can in fact be predicted. In contrast, the probabilistic model "knows" when it loses track of the state, and thus the error signal gets automatically "smoothed out". In practice, for the deterministic system the planning never finds trajectories with low expected cost, although inserting the policy from the probabilistic system yields a low cost. These local minima problems are very severe: In the cart-pole task, we have never managed to get the deterministic system to find a good solution, whereas the probabilistic system has never failed. Demos of the task with a deterministic dynamics model can also be found at `http://mlg.eng.cam.ac.uk/carl/ewrl08/`.

5.1 Current Limitations

Although our system works very well on the simple cart-pole problem, there are a number of ways in which the current implementation is limited:

Firstly, the system currently relies strictly on exploitation. When it has obtained a reasonable strategy it never veers from this. Thus, the strategy found is sometimes not close to an optimal strategy. For example, the experimental results reported in the previous section find a solution which swings left, then right and then left and up to balancing. In other equivalent runs, we have also seen a more direct strategy, such as left, then right up to balance. We never seem to find solutions worse than the one reported here, but there may be quite some sensitivity to the initial experience (which is random in our case). A principled solution allowing for explorations could be achieved within the current framework, by using a cost depending on both the expected cost and the variance of the cost. The variance of the cost can also be computed in closed form.

Secondly, our current approach learns very fast in terms of the amount of experience required to solve the task, but the computational demand is not negligible. In our current implementation, the optimization of the policy takes about 1 hour of CPU time. Performance can certainly be improved by writing more efficient code, and by speeding up the basic GP predictions using sparse approximations, for instance, the recent methods described in [13]. Nevertheless, it is not obvious that the scheme can necessarily learn in real time. However, once the policy has been learned, the computational requirements of applying the policy to a control task are trivial.

Thirdly, we have demonstrated learning in the special case where the observations are corrupted by only a very modest amount of observation noise. In principle, there is nothing to hinder the use of the algorithm when observations are very noisy, but one would probably have to estimate the current state more carefully—in the current setup we just assume that the state is measured exactly.

Finally, in an actual implementation, one would perhaps prefer a piecewise linear policy over the currently implemented piecewise constant policy. This could be achieved by formally treating the previously applied force as a component of the state.

6 Conclusions and Outlook

We have developed a framework based on flexible probabilistic non-parametric models for fast reinforcement learning in continuous state and action systems. The framework is conceptually simple, relying on well-established ideas, but a decisive difference is that we use fully probabilistic models of the world dynamics.

We have demonstrated the effectiveness of our approach on the cart-pole problem. Our algorithm finds a good policy from scratch using less than a minute worth of experiences—as far as we know this is an unprecedented speed for this kind of problem, which has previously been considered very hard to learn. Moreover, we require only very few assumptions: 1) we set the sampling time to be 200 ms, that is, somewhat shorter than the eigen-frequency of the system, 2) we set the trajectory length to be 5 s, that is, somewhat longer than the eigen-frequency and 3) we set the length-scale for the cost function to be $c = 0.25\,\text{m}$. We have not experimented with other settings, but believe that our system is fairly insensitive to these, as long as the order of magnitude is reasonable.

It is our belief that the success of our algorithm stems from the principled approach to handling the model's uncertainty. We anticipate that the effective solutions of more complex problems will also be possible using this algorithm. In the near future, we will explore how our algorithm performs on more challenging tasks, especially in systems with higher dimensional states. In particular, we believe that principled algorithms to address the exploration versus exploitation trade-off and other fundamental problems in practical algorithms for reinforcement learning will require careful quantification of model uncertainty.

Acknowledgements

We thank Jan Peters for many helpful discussions. MPD acknowledges support by the German Research Foundation (DFG) through grant RA 1030/1-3 to CER.

References

1. Sutton, R.S.: Integrated Architectures for Learning, Planning, and Reacting Based on Approximate Dynamic Programming. In: Proceedings of the Seventh International Conference on Machine Learning, pp. 215–224. Morgan Kaufman Publishers, San Francisco (1990)
2. Atkeson, C.G., Santamaría, J.C.: A Comparison of Direct and Model-Based Reinforcement Learning. In: Proceedings of the International Conference on Robotics and Automation (1997)
3. Atkeson, C.G., Schaal, S.: Robot Learning from Demonstration. In: Proceedings of the 14th International Conference on Machine Learning, Nashville, TN, USA, July 1997, pp. 12–20. Morgan Kaufmann, San Francisco (1997)
4. Abbeel, P., Quigley, M., Ng, A.Y.: Using Inaccurate Models in Reinforcement Learning. In: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, June 2006, pp. 1–8 (2006)
5. Poupart, P., Vlassis, N.: Model-based Bayesian Reinforcement Learning in Partially Observable Domains. In: Proceedings of the International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, FL, USA (January 2008)
6. Schaal, S.: Learning From Demonstration. In: Advances in Neural Information Processing Systems, vol. 9, pp. 1040–1046. The MIT Press, Cambridge (1997)
7. Abbeel, P., Ng, A.Y.: Exploration and Apprenticeship Learning in Reinforcement Learning. In: Proceedings of th 22nd International Conference on Machine Learning, Bonn, Germay, August 2005, pp. 1–8 (2005)
8. Peters, J., Schaal, S.: Learning to Control in Operational Space. The International Journal of Robotics Research 27(2), 197–212 (2008)
9. Kuss, M.: Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning. Ph.D thesis, Technische Universität Darmstadt, Germany (February 2006)
10. Rasmussen, C.E., Kuss, M.: Gaussian Processes in Reinforcement Learning. In: Advances in Neural Information Processing Systems, June 2004, vol. 16, pp. 751–759. The MIT Press, Cambridge (2004)
11. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning. The MIT Press, Cambridge (2006)
12. Girard, A., Rasmussen, C.E., Quiñonero Candela, J., Murray-Smith, R.: Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In: Advances in Neural Information Processing Systems, vol. 15, pp. 529–536. The MIT Press, Cambridge (2003)
13. Snelson, E., Ghahramani, Z.: Sparse Gaussian Processes using Pseudo-inputs. In: Advances in Neural Information Processing Systems, vol. 18, pp. 1257–1264. The MIT Press, Cambridge (2006)
14. Doya, K.: Reinforcement Learning in Continuous Time and Space. Neural Computation 12(1), 219–245 (2000)

United We Stand: Population Based Methods for Solving Unknown POMDPs

Noel Welsh and Jeremy Wyatt

School of Computer Science
The University of Birmingham
Birmingham B15 2TT UK

Abstract. Solving large unknown POMDPs is an open research problem. Policy search is one solution method that is attractive as it scales in the size of the policy, which is typically much simpler than the environment. We present a global search algorithm capable of finding good policies for POMDPs that are substantially larger than previously reported results. Our algorithm is general; we show it can be used with, and improves the performance of, existing local search techniques such as gradient ascent. Sharing information between the members of the population is the key to our algorithm and we show it results in better performance than equivalent parallel searches that do not share information. Unlike previous work our algorithm does not require the size of the policy to be known in advance.

1 Introduction

In this paper we address the problem of learning a finite-state controller (FSC) with an unknown number of internal states for an unknown partially observable Markov decision process (POMDP). Our method uses a population based search that shares information between individuals to concentrate the search within the most promising regions of the policy space. We show that this algorithm is:

– Capable of learning good policies for unknown POMDPs substantially larger than previously reported in the literature
– More efficient than using equivalent resources to search without sharing information
– Compatible with, and can improve upon the performance of existing local search techniques such as gradient ascent,

Furthermore, we do not require the size of the policy is known in advance.

POMDPs are a well known class of formal problems used to model a wide range of engineering tasks. If a POMDP is known planning algorithms may be used to solve it. Exact planning is intractable [5] and in practice exact methods can only solve problems with tens of states. Approximate methods are more scalable and can now provide solutions to problems with hundreds of states [9].

A more realistic and much harder problem is to solve an unknown POMDP, particularly when the number of underlying states is unknown. This is the most general POMDP problem, and the one we tackle here. The agent's task is to select actions to interact with the world, and learn an optimal policy based on the outcome of these actions. One approach to solving this problem is to build a model from interactions, and create a policy by solving the model. In many cases we have noticed that the optimal policy is much smaller than the model. Thus we search directly in the space of policies, skipping the model building step. This places our work in the area known as policy search[1, 8, 10].

Our initial experience with gradient ascent policy search showed it has a high variance due to noise in the gradient estimates. This work is motivated by a desire to reduce the variance by maintaining a population of policies. To focus our work on the effect of the population we have deliberately chosen a naive algorithm, with the expectation that more sophisticated algorithms will only improve the results. We show that sharing information within the population of policies is more efficient than equivalent searches without sharing, that this algorithm can solve POMDPs with hundreds of states, and that it can be combined with and improve the performance of existing policy search algorithms.

2 POMDPs and FSCs

Formally, we model the environment as a partially observable Markov decision process (POMDP), a tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{B} >$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $\mathcal{T}$ the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defining the probability $\mathcal{T}(s, a, s') = P(s'|a, s)$, ρ the reward function $\rho : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, $\mathcal{O}$ the set of output symbols, and $\mathcal{B}$ the observation function $\mathcal{B} : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ which defines the probability $\mathcal{B}(s, z) = P(z|s)$.

The agent is controlled by a policy, which we model as a finite state controller (FSC), a tuple $\mathcal{P} =< \mathcal{M}, \mu_a, \mu_m, m_0 >$ where $\mathcal{M}$ is the set of internal controller states, μ_a the action function $\mu_a : \mathcal{M} \times \mathcal{A} \rightarrow [0, 1]$ defining a probability distribution over all actions for each internal state, $P(a|m)$, μ_m the internal state transition function $\mu_m : \mathcal{M} \times \mathcal{O} \times \mathcal{M} \rightarrow [0, 1]$ defining a probability distribution over the next internal state given the current internal state and the current observation $P(m'|o, m)$, and m_0 the distribution over starting states $m_0 : \mathcal{M} \rightarrow [0, 1]$.

We represent a FSC as two matrices of positive real numbers: the action function of size $|\mathcal{M}| \times |\mathcal{A}|$, and the transition function of size $|\mathcal{M}| \times |\mathcal{O}| \times |\mathcal{M}|$. The action function represents the probability of performing an action given the state of the FSC. The transition function represents the probability of performing a state transition given the current state and observation.

The numbers in the state and transition functions are translated into probabilities by the Boltzmann (softmax) function. Let $\theta_i, i = 1 \ldots n$ be a set of positive real numbers. Then the Boltzmann function maps θ_i to a probability μ_i asμ_i=$exp(\theta_i/\tau)/\sum_j exp(\theta_j/\tau)$.

τ is the so-called temperature parameter that varies the sensitivity of the Boltzmann function to differences between values of θ. For all experiments τ was set to 1.

3 Policy Search Algorithm

We wish to focus on the value of sharing information between members of the population, so our search algorithm is deliberately naive to highlight this effect. The key components of our search algorithm are: the local search operators, which create a new policy from an existing one, the global search operator, which creates new policies from the population, and method of combining the operators. We now consider each aspect in turn.

3.1 Searching with a Single Finite State Controller

Recall we represent a FSC as two matrices of positive real numbers. The simplest operator generates a new policy from an existing one by randomly altering a value in one of these matrices. We must also consider changing the number of states, so we have another operator that creates random matrices bigger or smaller than the current policy. Finally, as we wish to show that our technique can used with existing policy search algorithms, we have reimplemented the GAPS gradient ascent algorithm[8], and use it as our third local search operator. In summary, they are:

- Random **perturbation** of the transition or action probabilities
- **Jumping** to a random FSC with one more or fewer states
- **GAPS** gradient ascent

The specifics of the operators are:

A perturbation changes a single randomly chosen value in either matrix. As the softmax involves an exponential the change cannot be too great or the resulting probabilities will be extremely skewed. For a value v the new value is given by $0.5v + uniformRandom(0.0, v)$.

A dimension jump creates a new policy with one more or one less state, within a defined maximum and minimum. The new policy is initialised with parameters uniformly drawn from the range $[0.1, 2]$.

See [8] for more information on GAPS.

3.2 Searching with a Population of Finite State Controllers

Our global search operator should use information from the entire population to generate a new policy. The simplest method we can conceive is to replace one member of the population with another. Thus our operator is:

- **Cloning** samples a FSC from the current population with probability proportional to estimated return.

3.3 Population Simulated Annealing with Information Sharing

Having chosen our search operators we now combine them in a search algorithm. The algorithm we choose is simulated annealing [4], which we modify to incorporate a population.

Simulated annealing is a stochastic hill climbing algorithm. The exploration/ exploitation tradeoff is controlled by a temperature parameter. At high temperatures up hill moves are only weakly favoured over down hill moves, while at low temperatures the algorithm approaches pure hill climbing. The temperature parameter is initially set high, to encourage exploration, and then gradually reduced towards zero using an algorithm known as the cooling schedule.

We call each temperature setting an iteration. For each iteration we generate proposed policies by applying the search operators to each member of the current population. Favouring simplicity, we select each operator with a fixed probability dependent on the experimental setup. The new population is constructed from the proposals as follows: A proposal is always added to the new population if its estimated return is greater than or equal to the estimated return of the policy from which it was generated. If this is not the case it is accepted with a probability inversely proportional to the difference in estimated return and proportional to the current temperature. This process continues until a specified number of proposals have been made. If no proposals are accepted in an iteration the search halts.

Given enough proposals and a slow enough cooling schedule simulated annealing is guaranteed to converge to the global optimum. This theoretical guarantee only holds for cooling schedules that are too slow to use in practice. Geometric cooling, where the next temperature is some fixed proportion of the current temperature, is commonly used though no guarantee of convergence holds in this case.

Our algorithm is given in Algorithm 1.

Algorithm 1: Population simulated annealing
Input: $temp$: the starting temperature; max: the maximum number of states, $proposals$: the number of proposed policies per temperature step, $size$: the number of individuals in the population.
Output: A population of optimised solutions

```
ANNEAL($min, max, proposals\ size$)
(1)   $p \leftarrow createInitialPopulation(size)$
(2)   while $temp > finalTemp$
(3)       $p' \leftarrow \emptyset$
(4)       foreach $s \leftarrow p$
(5)           for $i = 1$ to $proposals$
(6)               $proposed \leftarrow proposeNewSolution(s, p)$
(7)               $d \leftarrow quality(proposed) - quality(s)$
(8)               $a \leftarrow min(1, \frac{exp(d)}{temp})$
(9)               $u \leftarrow uniformRandom(0, 1.0)$
(10)              if $u < a$ then $s \leftarrow proposed$
(11)          $p' \leftarrow s \cup p'$
(12)      if $p = p'$ then return $(p)$
(13)      $p \leftarrow p'$
(14)      $temp \leftarrow decreaseTemperature(temp)$
(15)  return $p$
```

4 An Empirical Study

We ran our algorithm on five problems: Load/Unload, the M-Maze, the Small Maze, and the Tunnels Maze. The problems are summarised in Table 1.

Table 1. Summary of problems

Problem	States	Observations	Actions
Load/Unload	10	3	2
M-Maze	11	6	4
Small Maze	44	38	4
Tunnels	165	44	4

The Load/Unload problem is shown in Figure 1. The agent starts at the left end of a corridor five units long, and must move to the right end (load) and then return to the start (unload). The agent may only move left or right. Actions always succeed unless the action would take the agent into a wall in which case it stays where it is. The agent receives a reward of 1 when it loads (if it was not previously loaded), a reward of 1 when it unloads (if it was previously loaded), and a reward of -1 at all other times. Hence the maximum total (undiscounted) reward possible is -4.[1]

The optimal policy for Load/Unload requires two states (alternatively, one bit of memory) to record if the agent is loaded.

The M-Maze[6] is shown in Figure 2. The agent starts in either of the positions labelled `Start` and must navigate to the `Goal` position. The corner positions are the only means the agent has for distinguishing its starting place. The optimal policy for the M-Maze requires 4 states. The agent receives a reward of 1 when it reaches the goal and a reward of -1 at all other times, so the maximum total reward is -4.0.

The Small Maze is a 10x10 grid world, shown in Figure 3. The agent starts in the lower right corner and must navigate around obstacles to the upper left corner. We use two variants of the Small Maze: one with deterministic transitions, and one with stochastic transitions and two starting points. In the stochastic variant, actions have only a 0.875 probability of succeeding, and a 0.125 probability of keeping the agent in the same square. The Tunnels Maze, shown in Figure 4, has the same rules as the stochastic Small Maze with more states and observations.

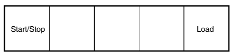


Fig. 1. The Load/Unload Task

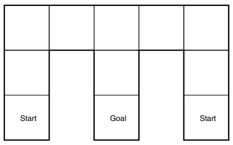


Fig. 2. The M-Maze

[1] This formulation of the Load/Unload problem follows [1]. This differs slightly from the formulation given in [8] as the agent receives a reward everytime it loads and unloads rather than only every time it unloads.

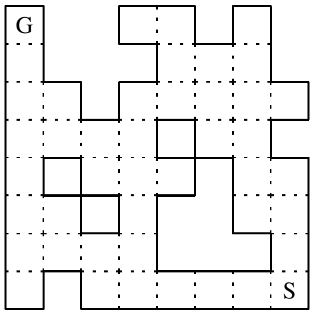


Fig. 3. The Small Maze

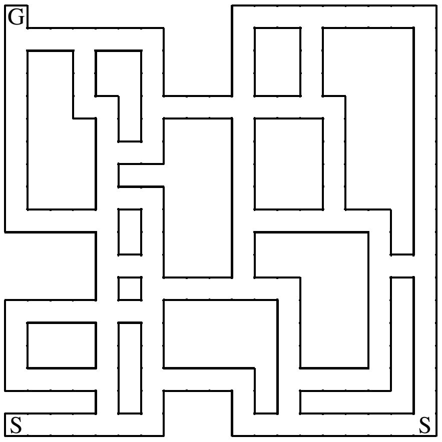


Fig. 4. The Tunnels Maze

The optimal policy for the deterministic Small Maze has a return of -12.0. To compute an upper bound on the return for the stochastic Small Maze we implemented an oracle optimal policy – that is, a policy with full knowledge of the true underlying state – and calculated an average return of -10.0 over 1000 runs. To establish a lower bound we ran 1000 randomly initialised polices with between one and five internal states, which had an average return of -681.3. For the Tunnels Maze the estimated upper bound -33.0 is and the lower bound is -917.8. It is important to note that the estimated upper bound is *not* obtainable by any policy within the search space. The optimal oracle policies have complete knowledge of the underlying state of the POMDP. Notice in particular that a FSC chooses an action based only on the current state of the FSC, so the first action taken by an FSC must be ignorant of the actual starting state. Hence an FSC cannot be optimal in either the Small Maze or the Tunnels Maze as both require different actions according to the randomly determined start state.

All problems have underlying deterministic transitions, except the stochastic Small Maze and the Tunnels Maze, and a significant degree of aliasing amongst observations. They all use a 'flat' representation; observations are encoded as numbers with no structure. For the Load/Unload and m-maze problems the observations indicate walls to the north, south, east, and west of the current position. For the other mazes observations indicate walls in the eight squares surrounding the current position. This is converted to a number by calculating all unique observations in the problem, and then assigning each observation a number. So, for example, in the Load/Unload problem each corner generates a unique observation, but the three corridor positions have the same observation. The observations are encoded as 1, 2, and 3 respectively.

5 Experimental Setup and Results

To validate the claims made for our algorithm we ran three groups of experiments:

To show the scalability of our algorithm we ran population simulated annealing on all the problems given in Section 4. The largest problem tackled by a model-free policy

Table 2. Results for scalability experiments. These experiments use perturbation, jumping, and cloning and demonstrate the effectiveness of the basic algorithm on a variety of problems large and small, and show the effect of varying the population size.

Pop.	Runs	Max	Min	μ	σ	Optimal
Load / Unload						
10	30	**-4.3**	-12.9	**-5.36**	1.57	*-4.0*
M-Maze						
10	39	**-4.0**	-14.4	-7.62	2.20	*-4.0*
20	39	-4.0	-12.2	-7.76	2.04	*-4.0*
40	40	-4.0	-10.8	-6.67	1.79	*-4.0*
80	7	-4.0	-8.4	**-5.91**	1.69	*-4.0*
Small Maze						
10	30	-13.0	-27.2	-17.52	3.39	*-12.0*
20	27	**-12.0**	-20.3	-15.80	3.22	*-12.0*
40	3	-12.5	-12.6	**-12.57**	0.06	*-12.0*
Stochastic Small Maze						
10	28	**-12.3**	-26.3	**-17.17**	3.34	*-10.0*
Tunnels						
10	30	**-38.4**	-101.3	**-57.41**	14.58	*-33.0*

Table 3. Results showing the effectiveness of information sharing. Experiments compare perturbation and jumping with and without cloning.

Variant	Pop.	Runs	Max	Min	μ	σ
Load / Unload						
Information sharing	10	30	**-4.3**	-12.9	**-5.36**	1.57
No information sharing	10	35	-4.9	-14.0	-10.09	3.33
M-Maze						
Information sharing	10	39	**-4.0**	-14.4	**-7.62**	2.20
No information sharing	10	30	-7.7	-13.7	-9.86	1.48

search method that we are aware of is the 52 state pentagon problem addressed by [1]. This is roughly the size of our Small Maze, and about one third the size of the Tunnels Maze. For Load/Unload, the M-Maze, and the Small Maze we also varied the population size to see how it affected performance. For these runs we used perturbation, jumping, and cloning. The results are shown in Table 2.

To show that sharing information between members of the population is an important contribution to the algorithms success we set the probability of cloning to zero for Load/Unload and the M-Maze, and compare these results to those obtained with cloning in our scalability experiments above. The results are collected in Table 3. The difference between both pairs of results is significant at the 0.05 level.

To show that population simulated annealing may be used with existing local search algorithms from the literature, we implemented the GAPS gradient ascent algorithm and then ran the following experiments:

Table 4. Results comparing GAPS without cloning, to GAPS and GAPS/perturbation hybrid with cloning. This demonstrates the information sharing algorithm improves existing policy search algorithms.

VARIANT	POP.	RUNS	MAX	MIN	μ	σ
M-MAZE						
GAPS, no information sharing	10	23	-4.0	-9.2	-6.25	1.49
GAPS	10	26	-4.0	-6.9	-5.14	0.81
GAPS and perturbation	10	25	-4.0	-6.8	**-4.69**	0.86

- We ran simulated annealing with GAPS on the M-Maze, with the probability of cloning set to zero. This establishes our baseline results. [2]
- We re-ran the above experiments with the probability of cloning set to 0.1, jumping set to 0.05, and probability of GAPS set to 0.85
- We re-ran the above with a hybrid local search: the probability of GAPS was set to 0.425 and the probability of perturbation was set to 0.425.

These results are shown in Table 4. The differences between GAPS with and without information sharing, and between GAPS with information sharing and the hybrid method are both significant at the 0.05 level.

The hybrid local search deserves some explanation. It has been shown that the initial gradient estimates for fully connected finite state controllers are often uninformative, and so gradient ascent may fail [1]. We expect this to occur in our experiments, and we hypothesise that the hybrid method will outperform pure GAPS by using perturbation to move to the controller to regions where the gradient is informative.

For all our experiments the initial temperature is 100.0, there are 1000 proposals per iteration, and return is estimated by averaging over 10 episodes.

For all runs the initial policy has one state, and parameters uniformly set to one. The number of states may vary between 1 and 5. Annealing is stopped when the temperature reaches 0.001 or if no new policy is accepted during an iteration. The cooling schedule is to set the next temperature to 0.9 of the current temperature (geometric cooling). Any episode that executes for more than 1000 steps without completing the task is halted.

Unless otherwise specified the probability of cloning is 0.10, the probability of jumping a dimension is 0.05, and the policy is perturbed otherwise.

6 Discussion and Related Work

We started this paper by claiming that our population based simulated annealing algorithm was:

- Capable of learning good policies for unknown POMDPs substantially larger than previously reported in the literature
- More efficient than using equivalent resources to search without sharing information

[2] The initial policy always has a single state. We maintain a non-zero probability of jumping to allow the search to vary the number of states.

– Compatible with, and can improve the performance of, existing local search techniques such as gradient ascent,

Our experiments have validated all three claims. Results on the Tunnels Maze show the algorithm is effective for POMDPs three times the size of previously reported results. The results with and without information sharing clearly show that sharing information within the population is a more efficient way to search, achieving a higher mean and lower variance. Our results with GAPS demonstrate that other policy search methods are compatible with, and improved by information sharing.

The hybrid GAPS/perturbation runs suggest that gradient information can be usefully combined with the random walks of perturbation to achieve performance better than either. The hybrid (or Hamiltonian) Monte Carlo method (see, e.g., [7] is a general framework for such approaches.

Results with varying population size show that increasing the population increases the mean and decreases the variance. This is unsurprising: the probability of finding a near optimal solution is directly proportional to amount of effort spent searching, which in turn is directly proportional to the population size. However, it is worth noting that our results suggest that a given amount of computing power is better invested in increasing the population size than increasing the number of runs with a given population size.

It is useful to inspect the learned policies to gain insight into the algorithm. Figure 5 presents a randomly chosen policy learned for the Tunnels Maze. The transition and emission probabilities have been omitted to reduce the visual complexity. The policy uses only two states, one for moving up, and another for moving laterally (and most of the time, moving left). The transitions between states are driven by a few significant observations. The optimal oracle policy for the Tunnels Maze requires more states than we allow our policies, however effective policies can still be found by generalising over the features of the problem; in the case of the Tunnels Maze this means tending to move left and up. However this particular case does not use the full memory available to, for example, differentiate between moving left and right.

The closest work to ours is the population based technique explored in [3]. They report results on 25x25 and 40x40 mazes, but their experimental setup is significantly different making comparison difficult. The key differences are in their policy representation. They use a factored representation for observations, and they represent a policy using a recurrent neural network, which has significantly different properties to a finite state controller.

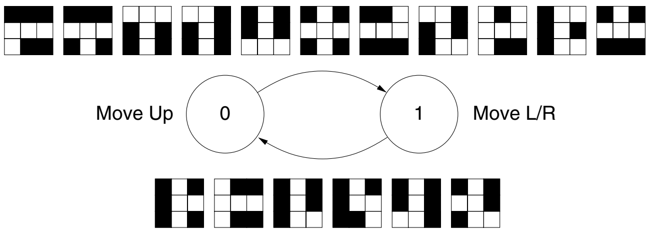


Fig. 5. An example FSC learned for the Tunnels Maze

Policy search ought to scale in the size of the policies rather than the size of the POMDP. In our algorithm we allow the size to vary so in theory the FSCs might grow very large. However, since each additional state leads to a quadratic increase in the number of parameters in the FSC exploring the space of larger models is correspondingly more difficult. This acts as a pressure to keep the size of the FSCs down. However it does mean performance could be poor if the problem has a large number of observations that are relevant to solving the problem. In these situations a factored representation will help to avoid explosion in the size of the search space.

The members of the population can be seen as samples from a distribution over the space of FSCs, and the algorithm as a mechanism for fitting that distribution to the areas with highest return. We might try to place an explicit distribution over the search space from which we sample. Preliminary experiments with a simple mixture model that chooses first a number of states, and then transmission and emission parameters showed that the number of states alone provides no useful information. An alternative is to use a Dirichlet process prior in the manner of the infinite HMM [2], allowing states to be shared amongst the population.

References

[1] Aberdeen, D.A.: Policy-Gradient Algorithms for Partially Observable Markov Decision Processes. Ph.D thesis, The Australian National University (2003)

[2] Beal, M., Ghahramani, Z., Rasmussen, C.E.: The infinite hidden Markov model. In: Advances in Neural Information Processing Systems, vol. 14, pp. 577–585. MIT Press, Cambridge (2002)

[3] Glickman, M.R., Sycara, K.: Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 194–201 (2001)

[4] Kirkpatrick Jr., S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)

[5] Littman, M.L.: Algorithms for Sequential Decision Making. Ph.D thesis, Brown University (1996)

[6] McCallum, A.: Reinforcement Learning with Selective Perception and Hidden State. Ph.D thesis, Department of Computer Science, University of Rochester (1995)

[7] Neal, R.M.: Probabilistic inference using Markov chain Monte Carlo methods. Technical report, Department of Computer Science, University of Toronto (1993)

[8] Peshkin, L., Meuleau, N., Kaelbling, L.P.: Learning policies with external memory. In: Proceedings of the Sixteenth International Conference on Machine Learning, pp. 307–314 (1999)

[9] Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1025–1032 (2003)

[10] Strens, M.J.A., Moore, A.W.: Direct policy search using paired statistical tests. In: Proc. 18th International Conf. on Machine Learning, pp. 545–552. Morgan Kaufmann, San Francisco (2001)

New Error Bounds for Approximations from Projected Linear Equations

Huizhen Yu[1] and Dimitri P. Bertsekas[2]

[1] Helsinki Institute for Information Technology (HIIT)
University of Helsinki, Finland
janey.yu@cs.helsinki.fi
[2] Laboratory for Information and Decision Systems (LIDS)
M.I.T., Cambridge, MA 02139, USA
dimitrib@mit.edu

Abstract. We consider linear fixed point equations and their approximations by projection on a low dimensional subspace. We derive new bounds on the approximation error of the solution, which are expressed in terms of low dimensional matrices and can be computed by simulation. When the fixed point mapping is a contraction, as is typically the case in Markovian decision processes (MDP), one of our bounds is always sharper than the standard worst case bounds, and another one is often sharper. Our bounds also apply to the non-contraction case, including policy evaluation in MDP with nonstandard projections that enhance exploration. There are no error bounds currently available for this case to our knowledge.

1 Introduction

For a given $n \times n$ matrix A and vector $b \in \Re^n$, let x^* and $\bar{x}$ be solutions of the two linear fixed point equations,

$$x = Ax + b\ , \qquad x = \Pi(Ax + b)\ , \tag{1}$$

respectively, where Π denotes projection on a k-dimensional subspace S with respect to certain weighted Euclidean norm $\|\cdot\|_\xi$. We assume that x^* and $\bar{x}$ exist, and that the matrix $I - \Pi A$ is invertible so that $\bar{x}$ is unique.

Implicit here is the assumption that n is very large, so that n-dimensional vector-matrix operations are practically impossible, while $k << n$. Our objective in solving the projected equation $x = \Pi(Ax+b)$ is to approximate the solution of the original equation $x = Ax + b$ using k-dimensional computations and storage. This approach is common in MDP, where A is a stochastic or substochastic matrix, and simulation-based approximate policy evaluation methods, based on temporal differences (TD), have been successfully used (see e.g., [2,8,9,10]). In our recent paper [3], we have extended these methods to the case where A is an arbitrary matrix, subject only to the restriction that $I - \Pi A$ is invertible.

In the MDP context, where ΠA is usually a contraction, there are two commonly used error bounds that compare the norms of $x^* - \bar{x}$ and $x^* - \Pi x^*$. The first bound (see e.g., [2,10]) holds if $\|\Pi A\| = \alpha < 1$ with respect to some norm $\|\cdot\|$, and has the form

$$\|x^* - \bar{x}\| \le \frac{1}{1-\alpha} \|x^* - \Pi x^*\| \ . \tag{2}$$

The second bound (see e.g., [11,1]) holds in the usual case where ΠA is a contraction with respect to the Euclidean norm $\|\cdot\|_\xi$, with ξ being the invariant distribution of the Markov chain underlying the problem, i.e., $\|\Pi A\|_\xi = \alpha < 1$. It is derived using the Pythagorean theorem $\|x^* - \bar{x}\|_\xi^2 = \|x^* - \Pi x^*\|_\xi^2 + \|\bar{x} - \Pi x^*\|_\xi^2$, and it is much sharper than the first bound:

$$\|x^* - \bar{x}\|_\xi \le \frac{1}{\sqrt{1-\alpha^2}} \|x^* - \Pi x^*\|_\xi \ . \tag{3}$$

The bounds (2), (3) are determined by the modulus of contraction α, and apply only when ΠA is a contraction mapping. We develop in this paper new error bounds, which are sharper when ΠA is a contraction, including important MDP cases, and also apply when ΠA is not a contraction.

Our starting point is the observation that the two terms involved in the bounds (2) and (3) satisfy the following equation with or without contraction assumptions:[1]

$$x^* - \bar{x} = (I - \Pi A)^{-1}(x^* - \Pi x^*) \ . \tag{4}$$

We may view the bounds (2), (3) as relaxed versions of this equation. In particular, we may obtain the bound (2) by writing $(I - \Pi A)^{-1} = I + \Pi A + \cdots$, and by upper-bounding each term in the expansion separately: $\|(\Pi A)^n\| \le \alpha^n$. We may obtain the bound (3) by writing

$$(I - \Pi A)^{-1} = I + \Pi A (I - \Pi A)^{-1} \ , \tag{5}$$

and by upper-bounding the norm of $\Pi A(I - \Pi A)^{-1}(x^* - \Pi x^*)$ by $\alpha \|x^* - \bar{x}\|_\xi$ and rearranging terms.[2] We will develop a different bounding approach, so that α will not be in the denominator of the bound. To this end, we will express $(I - \Pi A)^{-1}$ in the form

$$(I - \Pi A)^{-1} = I + (I - \Pi A)^{-1} \Pi A \ , \tag{6}$$

[1] This can be seen by subtracting $\bar{x} = \Pi(A\bar{x} + b)$ from $\Pi x^* = \Pi(Ax^* + b)$ to obtain

$$\Pi x^* - \bar{x} = \Pi A(x^* - \bar{x}) \ , \quad \Rightarrow \quad (\Pi x^* - x^*) + (x^* - \bar{x}) = \Pi A (x^* - \bar{x}) \ , \quad \Rightarrow \quad (4).$$

[2] From Eqs. (4)-(5) and the orthogonality of $(x^* - \Pi x^*)$ to the subspace S, we have

$$\begin{aligned} \|x^* - \bar{x}\|_\xi^2 &= \|x^* - \Pi x^*\|_\xi^2 + \|\Pi A (I - \Pi A)^{-1}(x^* - \Pi x^*)\|_\xi^2 \\ &= \|x^* - \Pi x^*\|_\xi^2 + \|\Pi A(x^* - \bar{x})\|_\xi^2 \le \|x^* - \Pi x^*\|_\xi^2 + \alpha^2 \|x^* - \bar{x}\|_\xi^2 \ . \end{aligned}$$

and aim at bounding the term $(I - \Pi A)^{-1} \Pi A(x^* - \Pi x^*)$ *directly* (this term is in fact $\Pi x^* - \bar{x}$, the bias of $\bar{x}$ from Πx^*). In doing so, we will obtain bounds that not only can be sharper than the preceding bounds for the contraction case, but also carry over to the non-contraction case.

We will derive two bounds, which involve the spectral radii of small-size matrices, and provide a "data/problem-dependent" error analysis, in contrast to the fixed error bounds (2), (3); see Theorems 1 and 2. The bounds are *independent* of the parametrization of the subspace S, and can be computed with low-dimensional operations and simulation, if this is desirable. One of the bounds is sharper than the other, but involves more complex computations. We also have some additional bounds that provide insight into the character of the approximation error, but are qualitative in nature; they are given in an extended version of this paper [12].

Our bounds have the general form

$$\|x^* - \bar{x}\|_\xi \leq B(A, \xi, S) \, \|x^* - \Pi x^*\|_\xi \,, \tag{7}$$

where $B(A, \xi, S)$ is a constant that depends on A, ξ, and S (but not on b). Like the bounds (2), (3), we may view $\|x^* - \Pi x^*\|_\xi$ as the *baseline error*, i.e., the minimum error in estimating x^* by a vector in the approximation subspace S. We may view $B(A, \xi, S)$ as an upper bound to the *amplification ratio*, $\|x^* - \bar{x}\|_\xi / \|x^* - \Pi x^*\|_\xi$, which is due to solving the projected equation $x = \Pi(Ax + b)$ instead of projecting x^* on S (or equivalently, view $\sqrt{B^2(A, \xi, S) - 1}$ as an upper bound to the "bias-to-distance" ratio $\|\bar{x} - \Pi x^*\|_\xi / \|x^* - \Pi x^*\|_\xi$).

We present our main results in the next section. In Section 3, we address the application of the new error bounds to the approximate policy evaluation in MDP and to the far more general problem of approximate solution of large systems of linear equations. Due to space limitation, proofs and additional related analysis are omitted; they can be found in the expanded version of the present paper [12].

2 Main Results

We first introduce the main theorems and explain the underlying ideas. Let Φ be an $n \times k$ matrix whose columns form a basis of S. Let Ξ be a diagonal matrix with ξ on the diagonal. Define $k \times k$ matrices B, M, and F by

$$B = \Phi' \Xi \Phi \,, \qquad M = \Phi' \Xi A \Phi \,, \qquad F = (I - B^{-1} M)^{-1} \tag{8}$$

(we will show later that the inverse in the definition of F exists). Notice that the projection matrix Π can be expressed as $\Pi = \Phi(\Phi' \Xi \Phi)^{-1} \Phi' \Xi = \Phi B^{-1} \Phi' \Xi$. For a square matrix L, let $\sigma(L)$ denote the spectral radius of L.

Theorem 1. *The approximation error $x^* - \bar{x}$ satisfies*

$$\|x^* - \bar{x}\|_\xi \leq \sqrt{1 + \sigma(G_1) \|A\|_\xi^2} \; \|x^* - \Pi x^*\|_\xi \,, \tag{9}$$

where G_1 is the $k \times k$ matrix

$$G_1 = B^{-1}F'BF \; . \tag{10}$$

Furthermore, $\sigma(G_1) = \|(I - \Pi A)^{-1}\Pi\|_\xi^2$, so the bound (9) is invariant to the choice of basis vectors of S (i.e., Φ).

The idea in deriving Theorem 1 is to combine Eqs. (4)-(5) with the bound

$$\left\|(I - \Pi A)^{-1}\Pi A(x^* - \Pi x^*)\right\|_\xi \leq \left\|(I - \Pi A)^{-1}\Pi\right\|_\xi \|A\|_\xi \left\|x^* - \Pi x^*\right\|_\xi \; ,$$

and to show that $\|(I - \Pi A)^{-1}\Pi\|_\xi^2 = \sigma(G_1)$. An important fact, to be demonstrated later, is that G_1 can be obtained by simulation, using low dimensional calculations.

While the bound of Theorem 1 can be conveniently computed, it is less sharp than the bound of the subsequent Theorem 2, and under certain circumstances less sharp than the bound (3). In Theorem 1, $\|A\|_\xi$ is needed, and this can be a drawback, particularly for the non-contraction case. In Theorem 2, $\|A\|_\xi$ is no longer needed; A is absorbed into the matrix to be estimated. Furthermore, Theorem 2 takes into account that $x^* - \Pi x^*$ is perpendicular to the subspace S; this considerably sharpens the bound. On the other hand, the sharpened bound of Theorem 2 involves a $k \times k$ matrix R (defined below) in addition to B and M, which may not be straightforward to estimate in some cases, as will be commented later.

Theorem 2. *The approximation error $x^* - \bar{x}$ satisfies*

$$\|x^* - \bar{x}\|_\xi \; \leq \; \sqrt{1 + \sigma(G_2)} \; \|x^* - \Pi x^*\|_\xi \; , \tag{11}$$

where G_2 is the $k \times k$ matrix

$$G_2 = B^{-1}F'BFB^{-1}(R - MB^{-1}M') \; , \tag{12}$$

and R is the $k \times k$ matrix

$$R = \Phi'\Xi A \Xi^{-1}A'\Xi\Phi \; .$$

Furthermore, $\sigma(G_2) = \|(I - \Pi A)^{-1}\Pi A(I - \Pi)\|_\xi^2$, so the bound (11) is invariant to the choice of basis vectors of S (i.e., Φ).

The idea in deriving Theorem 2 is to combine Eqs. (4)-(5) with the bound

$$\begin{aligned}\left\|(I - \Pi A)^{-1}\Pi A(x^* - \Pi x^*)\right\|_\xi &= \left\|(I - \Pi A)^{-1}\Pi A(I - \Pi)(x^* - \Pi x^*)\right\|_\xi \\ &\leq \left\|(I - \Pi A)^{-1}\Pi A(I - \Pi)\right\|_\xi \left\|x^* - \Pi x^*\right\|_\xi \; ,\end{aligned}$$

and to show that $\|(I - \Pi A)^{-1}\Pi A(I - \Pi)\|_\xi^2 = \sigma(G_2)$. Incorporating the matrix $I - \Pi$ in the definition of G_2 is crucial for improving the bound of Theorem 1.

Estimating the matrix R, although not always as straightforward as estimating B and M, can be done for a number of applications. A primary exception is when A itself is an infinite sum of powers of matrices, which is the case of the TD(λ) method with $\lambda > 0$. We will address these issues in Section 2.3.

2.1 Key Arguments for Proofs

Due to space limitation, we omit the proofs and only point out the main proof arguments. We shall need two technical lemmas. The first lemma introduces an expression of the matrix $(I-\Pi A)^{-1}$ that will be used to derive our error bounds. The second lemma establishes the relation between the norm of an $n \times n$ matrix that is a product of $n \times k$ and $k \times n$ matrices, and the spectral radius of a certain product of $k \times k$ matrices.

Lemma 1. *The matrix $I - \Pi A$ is invertible if and only if the inverse $(I - B^{-1}M)^{-1}$ defining F exists. When $I - \Pi A$ is invertible, $(I - \Pi A)^{-1}$ maps S onto S, and furthermore,*

$$(I - \Pi A)^{-1} = I + (I - \Pi A)^{-1}\Pi A = I + \Phi F B^{-1}\Phi' \Xi A \,. \qquad (13)$$

Note that since B and M are low-dimensional matrices, the first part of Lemma 1 is useful for verifying the existence of the inverse of $I - \Pi A$ using the data.

Lemma 2. *Let H and D be an $n \times k$ and $k \times n$ matrix, respectively. Let $\|\cdot\|$ denote the standard (unweighted) Euclidean norm. Then,*

$$\|HD\|_\xi^2 = \|\Xi^{1/2} H D \Xi^{-1/2}\|^2 = \sigma\big((H'\Xi H)(D\Xi^{-1}D')\big) \,. \qquad (14)$$

Theorems 1 and 2 can now be proved by combining Lemmas 1 and 2, the relation (4), and the proof ideas described immediately after the statements of the theorems.

2.2 Comparison of Error Bounds

The error bounds of Theorems 1 and 2 apply to the general case where ΠA is not necessarily a contraction mapping, while the worst case error bounds (2) and (3) only apply when ΠA is a contraction. We will thus compare them for the contraction case. Nevertheless, our discussion will illuminate the strengths and weaknesses of the new bounds for both contraction and non-contraction cases.

First it can be shown that the error bound of Theorem 2 is always the sharpest.

Proposition 1. *Assume that $\|\Pi A\|_\xi \leq \alpha < 1$. Then, the error bound of Theorem 2 is always no worse than the error bound (3), i.e., $1+\sigma(G_2) \leq 1/(1-\alpha^2)$, where G_2 is given by Eq. (12).*

It can also be shown that the error bound of Theorem 2 is tight in the sense that there is a worst case choice of b for which the bound holds with equality.

Let us compare now the error bound of Theorem 1 with the bounds (2) and (3) from the worst case viewpoint. Since Theorem 1 is effectively equivalent to

$$\big\|(I - \Pi A)^{-1}\Pi A(x^* - \Pi x^*)\big\|_\xi \leq \big\|(I - \Pi A)^{-1}\Pi\|_\xi \|A\|_\xi \|x^* - \Pi x^*\big\|_\xi \ ,$$

we see that the bound of Theorem 1 is never worse than the bound (2), because we have bounded the norm of the matrix $(I - \Pi A)^{-1}\Pi$ as a whole, instead of bounding each term in its expansion separately as in the case in the bound (2). However, the bound of Theorem 1 can be degraded by two over-relaxations:

(i) The residual vector $x^* - \Pi x^*$ is special, in that it satisfies $\Pi(x^* - \Pi x^*) = 0$, but the bound does not use this fact.
(ii) When ΠA is zero or near zero, the bound cannot fully utilize this fact.

The effect of (i) can be quite significant when A has a dominant real eigenvalue β with an eigenvector x that lies in the approximation subspace S. In such a case, the bound reduces essentially to the bound (2), since

$$\|(I - \Pi A)^{-1}\Pi x\|_\xi = \tfrac{1}{1-\beta}\|x\|_\xi \, . \tag{15}$$

This happens because the analysis has not taken into account that the residual vector $(x^* - \Pi x^*)$ cannot be an eigenvector that is contained in S.

The relaxation related to (ii) may not look obvious in the current analysis; it does, however, in an alternative equivalent form of the analysis, by noticing that

$$(I - \Pi A)^{-1}\Pi A = \Pi A + \Pi A(I - \Pi A)^{-1}\Pi A \, , \tag{16}$$

and the norm of the matrix on the right has been bounded by $\|\Pi + \Pi A(I - \Pi A)^{-1}\Pi\|_\xi\|A\|_\xi$ in Theorem 1. When $\Pi A = 0$ the matrix of Eq. (16) is zero but its bound is not, because the matrices Π and A are split in the bounding procedure. Accordingly, the spectral radius $\sigma(G_1)$ becomes $\|\Pi\|_\xi^2 = 1$. Similarly, over-relaxation occurs when ΠA is not zero but is near zero.[3]

The two shortcomings of the bound of Theorem 1 arise in the MDP applications that we will discuss, as well as in non-contraction cases. On the other hand, there are cases where Theorem 1 provides sharper bounds than the fixed error bound (3), and cases where Theorem 1 gives computable bounds while the bound (3) is qualitative (for example, when the modulus of contraction of ΠA is unknown). We also mention that the expanded version of this paper [12] contains additional analysis, which in part addresses the shortcomings just discussed.

The advantage that the bound of Theorem 1 holds over the one of Theorem 2 is that it is rather easy to compute: the matrices B and M define the solution $\bar{x}$, so the bound is obtained together with the approximating solution without extra computation overhead. By contrast, the bound of Theorem 2 involves the matrix R, which can be hard to estimate for certain applications.

2.3 Estimating the Low Dimensional Matrices in the Bounds

We consider estimating the $k \times k$ matrices involved in the bounds by simulation, and we focus on estimating the matrix R in Theorem 2:

$$R = \Phi' \Xi A \Xi^{-1} A' \Xi \Phi \, .$$

Other cases do not seem to need explanations: the estimation of B and M using simulation has been well explained in the literature (see eg., [3,4,7]); and if instead of using simulation, products of $k \times n$ and $n \times n$ matrices can be

[3] In practice, when using the bound of Theorem 1, one may check if ΠA is near zero by checking if M is.

computed directly, then the calculation of R may be done directly with common matrix algebra.

First, let us note that when the matrix Φ actually used in the simulation does not have full rank, Theorems 1 and 2 imply that the bounds can be computed by using the pseudo-inverse of B, neglecting zero eigenvalues (a tolerance level/threshold needs to be determined, of course, in the simulation context).

Without loss of generality, in this subsection, we assume that $\sum_{i=1}^{n} \xi_i = 1$ so that ξ can be viewed as a distribution. In practice, we never need to normalize ξ as the normalization constant will be canceled in the product defining the matrices G_1 and G_2. Let $\phi(i)'$ denote the i-th row of Φ. Our methods for estimating R are based on a common procedure: we first express R as a summation of $k \times k$ matrices, e.g.,

$$R = \sum_{i,j,\hat{j}} (a_{ji} a_{\hat{j}i}) \cdot \tfrac{\xi_j \xi_{\hat{j}}}{\xi_i} \cdot \phi(j)\phi(\hat{j})' \ ,$$

and guided by this expression, we generate samples and choose proper weights for them, so that each term in the summation is matched by a weighted long-run average of respective samples.

We will give four examples that apply to different contexts, depending on whether the entries of ξ and A in the preceding formula for R are explicitly known or not, with two main applications in our mind:

(i) *General linear equations* in which we know explicitly the entries of A, and we may want to choose a particular projection norm, for instance, the standard Euclidean norm (all entries of ξ being equal). The procedure of Example 1 and its slight variant in Example 2 refer primarily to this case.

(ii) *Markov decision processes* in which we do not know A, but we can generate samples by simulation of a certain Markov chain underlying the problem. Examples 3 and 4 are mostly relevant to this case, including in particular, evaluating the cost or Q-factors of a policy using TD(0)-like algorithms, with and without exploration enhancements. (We refer to our paper [3] for some algorithms involving exploration, where the simulation procedures of Examples 3 and 4 may apply.)

Example 1. Both ξ and A are known explicitly. We express R as the summation given above and generate a sequence of triple indices $(i_t, j_t, \hat{j}_t)$ as follows. We generate the sequence $(i_0, i_1, \ldots)$ so that its empirical distribution converges to ξ. At i_t, we generate two mutually independent transitions (i_t, j_t) and $(i_t, \hat{j}_t)$ according to a certain transition probability matrix P with $p_{ij} \neq 0$ whenever $a_{ji} \neq 0$. We then define R_t by

$$R_t = \frac{1}{t+1} \sum_{m=0}^{t} \left(\tfrac{a_{j_m i_m}}{p_{i_m j_m}} \cdot \tfrac{a_{\hat{j}_m i_m}}{p_{i_m \hat{j}_m}} \right) \cdot \tfrac{\xi_{j_m} \xi_{\hat{j}_m}}{\xi_{i_m}^2} \cdot \phi(j_m)\phi(\hat{j}_m)' \ ,$$

where t is a suitably large number, and approximate R by the symmetrized matrix $(R_t + R_t')/2$. Note that in the special case where $\Xi = \frac{1}{n} I$, the indices

i_t can be generated independently with the uniform distribution, R reduces to $\frac{1}{n}\Phi' A A' \Phi$, and the ratio $\frac{\xi_{j_m}\xi_{\hat{j}_m}}{\xi^2_{i_m}}$ in R_t reduces to 1. □

Example 2. The weight vector ξ is not known explicitly, but A is; moreover, a sequence $(i_0, i_1, \ldots)$ can be generated so that its empirical distribution converges to ξ. For example, ξ may be the unique invariant distribution of a Markov chain, which is used to generate the sequence $(i_0, i_1, \ldots)$. In this case, we can keep tracking the empirical distribution $\hat{\xi}_t$ of the sequence i_t up to time t. We then apply the same sampling and estimation schemes as in Example 1, replacing the ratio $\frac{\xi_{j_m}\xi_{\hat{j}_m}}{\xi^2_{i_m}}$ in R_t by $\frac{\hat{\xi}_{t,j_m}\hat{\xi}_{t,\hat{j}_m}}{\hat{\xi}^2_{t,i_m}}$. □

Example 3. Both ξ and A are not known explicitly; moreover, the ratios $\beta_{ij} = a_{ij}/p_{ij}$ are known for a certain transition matrix P with $p_{ij} \neq 0$ whenever $a_{ij} \neq 0$, and ξ is the unique invariant distribution of the Markov chain associated with P. While P is not explicitly known, it is assumed that a simulator is available that can generate transitions according to P.

To estimate R, we first express it as

$$R = \sum_{i,l,j} (\beta_{il}\beta_{jl}) \cdot \left(\xi_i p_{il} \cdot \tfrac{p_{jl}\xi_j}{\xi_l} \right) \cdot \phi(i)\phi(j)' \,.$$

Noticing that $\frac{p_{jl}\xi_j}{\xi_l}$ equals the steady-state conditional probability $P(X_{t-1} = j \mid X_t = l)$ for the Markov chain X_t, we thus generate a sequence of pairs of indices (i_t, j_t) as follows. Let $(i_0, i_1, \ldots)$ be a trajectory of the Markov chain. At $i_{t+1} = l$, we generate, using the uniform distribution, one sample (j, l) from the set of past transitions to l, $\{(i_{t_k-1}, i_{t_k}) \mid i_{t_k} = l, t_k \leq t+1\}$, and we let $j_t = j$. (Indeed, this will also work if we simply let $j_t = i_{t_{k-1}}$ where t_k is the most recent time prior to $t+1$ that $i_{t_k} = l$.) It can be seen that the conditional probability of j_t given i_{t+1} converges asymptotically to $\frac{p_{j_t i_{t+1}}\xi_{j_t}}{\xi_{i_{t+1}}}$. We then define R_t by

$$R_t = \frac{1}{t+1} \sum_{m=0}^{t} (\beta_{i_m i_{m+1}} \beta_{j_m i_{m+1}}) \cdot \phi(i_m)\phi(j_m)' \,,$$

and we approximate R by the symmetrized matrix $(R_t + R_t')/2$.

If the Markov chain is reversible, i.e., $\xi_j p_{jl} = \xi_l p_{lj}$ for all j, l, then the method can be substantially simplified. We can omit the procedure of generating j_t and simply set $j_m = i_{m+2}$ in R_t, because if we do so, the proper weight for the sample is $\frac{\xi_{j_m} p_{j_m i_{m+1}}}{\xi_{i_{m+1}} p_{i_{m+1} j_m}} = 1$. □

Example 4. The weight vector ξ is known explicitly, but A is not; moreover, the ratios $\beta_{ij} = a_{ij}/p_{ij}$ are known for a certain transition matrix P with $p_{ij} \neq 0$ whenever $a_{ij} \neq 0$. Here, ξ need not be the invariant distribution of P.

We can deal with this case by combining partially the schemes in Examples 2 and 3. We express R and generate a sequence of pairs of indices (i_t, j_t) as in

Example 3. We keep tracking the empirical distribution κ_t of the sequence i_t up to time t, to approximate the invariant distribution of P. We weight samples properly to define R_t:

$$R_t = \frac{1}{t+1} \sum_{m=0}^{t} (\beta_{i_m i_{m+1}} \beta_{j_m i_{m+1}}) \cdot \left(\frac{\xi_{i_m} \xi_{j_m}}{\xi_{i_{m+1}}} \cdot \frac{\kappa_{t,i_{m+1}}}{\kappa_{t,i_m} \kappa_{t,j_m}} \right) \cdot \phi(i_m)\phi(j_m)' \ ,$$

and we approximate R by the symmetrized matrix $(R_t + R_t')/2$.

If the Markov chain associated with P is reversible, then there is simplification, similar to that in Example 3. We simply set $j_t = i_{t+2}$ and

$$R_t = \frac{1}{t+1} \sum_{m=0}^{t} (\beta_{i_m i_{m+1}} \beta_{i_{m+2} i_{m+1}}) \cdot \left(\frac{\xi_{i_m} \xi_{i_{m+2}}}{\xi_{i_{m+1}}} \cdot \frac{\kappa_{t,i_{m+1}}}{\kappa_{t,i_m} \kappa_{t,i_{m+2}}} \right) \cdot \phi(i_m)\phi(i_{m+2})' \ ,$$

because the extra term needed for weighting the sample properly is $\frac{\kappa_{t,j_m} p_{j_m i_{m+1}}}{\kappa_{t,i_{m+1}} p_{i_{m+1} j_m}}$, which converges to 1 as $m \to \infty$. □

A main source of difficulty in the estimation of R in MDP, as Examples 3 and 4 illustrate, is the unknown matrix A and the need of samples of "backward" transitions from a common state/index. Simulating backward transitions according to the steady-state conditional distribution is in general not easy. Consistently, as Example 1 illustrates, the estimation of R is quite simple when backward transitions can be easily generated, such as when A is known. A second source of difficulty in the estimation of R, as Examples 2-4 illustrate, is the memory demand. In particular, in order to either generate backward transitions or to weight samples properly, we must keep track of the past history of the simulation (except in the case of Example 3 and a reversible Markov chain).

Another drawback of the procedures given in Examples 1-4 is that they do not adapt easily to the case where A itself is a summation of infinitely many matrices, as in TD(λ) with $\lambda > 0$.

3 Applications

We consider two applications of Theorems 1 and 2. The first one is cost function approximation in MDP with TD-type methods. This includes single policy evaluation with discounted and undiscounted cost criteria. The second application is approximately solving large general systems of linear equations. We also illustrate with figures various issues discussed in Section 2.2 on the comparison of the bounds. We note that for TD(λ) with $\lambda > 0$, we do not yet have an efficient simulation-based method for estimating the bound of Theorem 2; we have calculated the bound using common matrix algebra, and we plot it just for comparison.

3.1 Cost Function Approximation for MDP

For policy evaluation in MDP, x^* is the cost function of the policy to be evaluated. Let P be the transition matrix of the Markov chain induced by the policy.

The original linear equation that we want to solve is the Bellman equation, or optimality equation, satisfied by x^*. It takes the form

$$x^* = b + \alpha P x^* ,$$

where b is the per-stage cost vector, and $\alpha \in [0, 1]$ is the discount factor: $\alpha \in [0, 1)$ corresponds to the discounted cost criterion, while $\alpha = 1$ corresponds to either the total cost criterion or the average cost criterion (in the latter case b is the per-stage cost minus the average cost). For simplicity of discussion, we assume that the Markov chain is irreducible.

With the TD(λ) method, we solve a projected form of the multistep Bellman equation $x = \Pi b + \Pi A x$, where A is defined for a pair of values (α, λ) by

$$A = P^{(\alpha,\lambda)} \overset{def}{=} (1-\lambda) \sum_{l=0}^{\infty} \lambda^l (\alpha P)^{l+1}$$

with either $\alpha \in [0, 1), \lambda \in [0, 1]$, or $\alpha = 1, \lambda \in [0, 1)$. Notice that the case $\lambda = 0$ corresponds to $A = \alpha P$.

Discounted Problems. Consider the discounted case: $\alpha < 1$. For $\lambda \in [0, 1]$, with ξ being the invariant distribution of the Markov chain, the modulus of contraction of $P^{(\alpha,\lambda)}$ with respect to $\| \cdot \|_\xi$ is

$$\|P^{(\alpha,\lambda)}\|_\xi = \frac{(1-\lambda)\alpha}{1-\lambda\alpha} .$$

Let e denote the constant vector of all ones. Like P, the matrix $P^{(\alpha,\lambda)}$ has e as an eigenvector associated with the dominant eigenvalue $\frac{(1-\lambda)\alpha}{1-\lambda\alpha}$.

If the approximation subspace S contains or nearly contains e, the bound of Theorem 1 can degrade to the worst case error bound given by (2), as remarked in Section 2.2. In such a case, in order to have a sharper bound for the approximation of Πx^*, we can estimate separately the projection of x^* on e and the projection of x^* on another subspace $\widehat{S} = (S \oplus e) \cap e^\perp$, which is the orthogonal complement of e in $S \oplus e$ (see Figure 1), and redefine $\bar{x}$ as the sum of the two estimates. It can be shown that when the first projection can be estimated with no bias, the error bound for the second projection carries over to the combined estimate $\bar{x}$.[4] Fortunately, for MDP, the projection of x^* on e can be calculated

[4] For a subspace V, let Π_V denote the projection on V. Let V and W be two orthogonal subspaces with $\Pi_V x^*$ known. Since $x^* - \Pi_V x^*$ satisfies the linear equation $x = Ax + \tilde{b}$ with $\tilde{b} = b + A\Pi_V x^* - \Pi_V x^*$, to obtain an estimate of $\Pi_W x^* = \Pi_W(x^* - \Pi_V x^*)$, we can solve the projected equation $x = \Pi_W A x + \Pi_W \tilde{b}$. (In the above MDP case, V is an eigenspace of A, so $\tilde{b}$ can be replaced by b.) Denote the solution by $\bar{x}_w$. Then, error bounds for $\bar{x}_w$ that are of the form $\|(x^* - \Pi_V x^*) - \bar{x}_w\|_\xi \leq L\|(x^* - \Pi_V x^*) - \Pi_W(x^* - \Pi_V x^*)\|_\xi$, are equivalent to error bounds $\|x^* - \bar{x}\|_\xi \leq L\|x^* - \Pi_{V\oplus W} x^*\|_\xi$ with $\bar{x} = \Pi_V x^* + \bar{x}_w$.

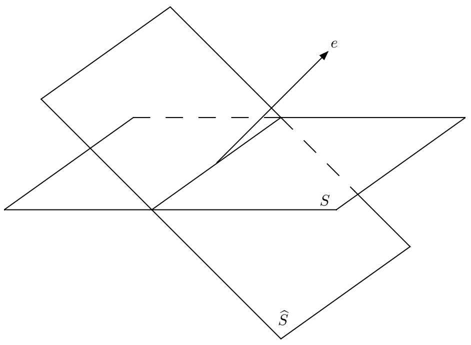


Fig. 1. Illustration of $\widehat{S}$, the orthogonal complement of e in $S \oplus e$, i.e., $\widehat{S} = (S \oplus e) \cap e^{\perp}$

asymptotically exactly through simulation.[5] In addition, basis vectors of $\widehat{S}$ can also be generated from Φ by using simulation (see eg., [5]), along with the approximation of the matrices B and M and without incurring much computation overhead. Figure 2 illustrates the error bounds, and shows how the use of $\widehat{S}$ may improve them. It can be observed that the bound of Theorem 2 has consistently performed best, as indicated by the analysis.

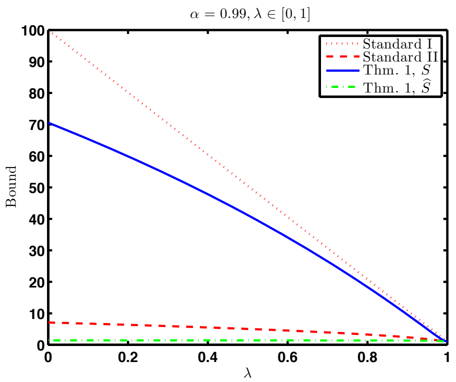
Figure 3 compares the bounds for the case where the projection norm is the standard unweighted Euclidean norm. The standard bounds and the bound of Theorem 1 need the value $\|A\|$, while the bound of Theorem 2 does not. For comparison of these bounds, we compute $\|P\|$ using the knowledge of P, bound $\|A\|$ by $\frac{(1-\lambda)\|\alpha P\|}{1-\lambda\|\alpha P\|}$, and plug the latter in the standard bounds and the bound of Theorem 1. The value $\|\alpha P\|$, which corresponds to $\|A\|$ for $\lambda = 0$, is shown in the titles of Figure 3. With the norm being different from $\|\cdot\|_{\xi}$, the mapping ΠA is not necessarily a contraction for small values of λ, even though in this example it is.

Note that the availability of computable error bounds for non-contraction mappings facilitates the design of policy evaluation algorithms with improved exploration. In particular, we can use the LSTD algorithm [4] to evaluate the cost or the Q-factor of a policy using special sampling methods that enhance exploration, and use the bound of Theorem 1 to estimate the corresponding amplification ratio.[6] Alternatively, we may use the bound of Theorem 2 in conjunction with TD(0)-type algorithms. Examples 3 and 4 show how to estimate the matrix R in cases where the projection norm is determined by an exploration policy, and where the projection norm is given explicitly with the desirable weights, respectively.

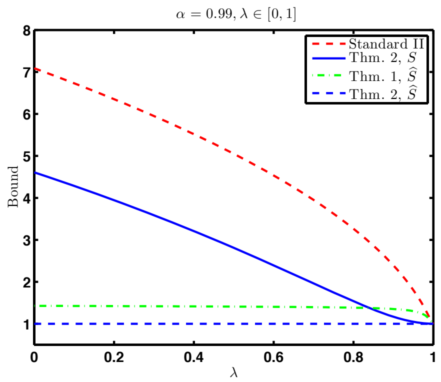
[5] It can be seen that the projection of x^* on e equals

$$\xi' x^* = \xi' b + \xi' P^{(\alpha,\lambda)} x^* = \xi' b + \frac{(1-\lambda)\alpha}{1-\lambda\alpha} \xi' x^*, \quad \Rightarrow \quad \xi' x^* = \frac{1-\lambda\alpha}{1-\alpha} \xi' b \,.$$

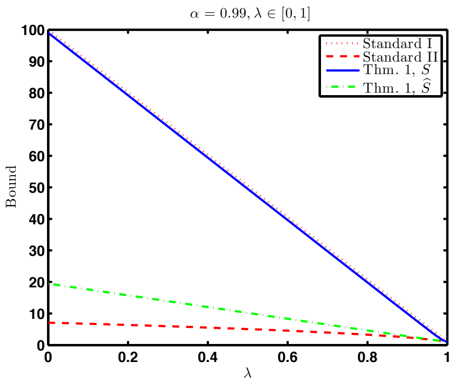
[6] When ΠA is not necessarily a contraction, a bound on $\|A\|_{\xi}$ is needed to apply Theorem 1. There are also algorithms involving exploration and maintaining the contraction property of ΠA, for which we refer to our paper [3].



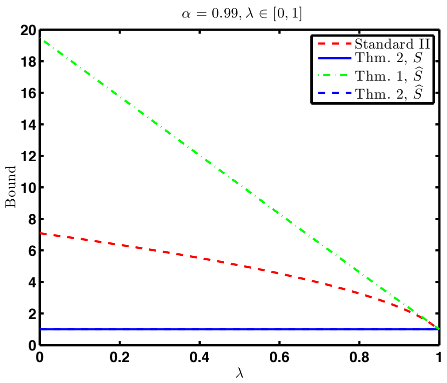
(a) Standard bounds vs. Theorem 1



(b) Standard bounds vs. Theorems 1 & 2 [detail of lower portion of (a)]



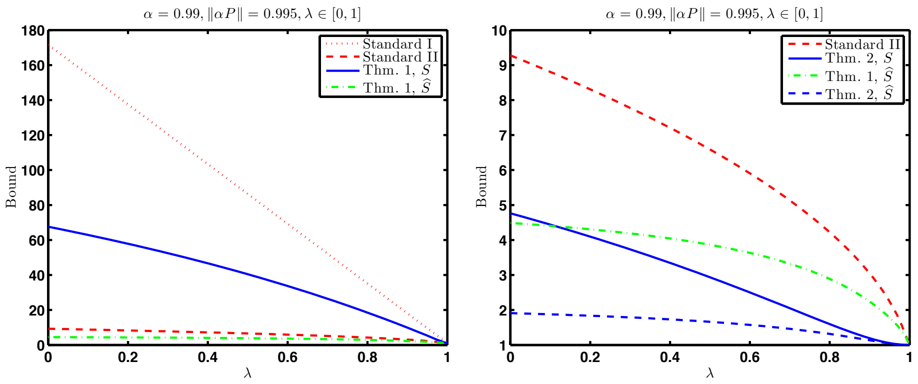
(c) Standard bounds vs. Theorem 1



(d) Standard bounds vs. Theorems 1 & 2 [detail of lower portion of (a)]

Fig. 2. Comparison of error bounds as functions of λ for two discounted problems with randomly generated Markov chains. The dimension parameters are $n = 200, k = 50$, and the weights ξ in the projection norm is the invariant distribution. Standard I and II refer to the worst case bounds (2) and (3), respectively. The Markov chain is the same in (a) and (b), and in (c) and (d). In (c) and (d), the Markov chain has a "noisy" block structure with two blocks, thus P has a relatively large subdominant eigenvalue; S is chosen to contain e and a vector close to an eigenvector associated with that subdominant eigenvalue. The subspace $\widehat{S}$ is derived from S by orthogonalization, as shown in Figure 1.

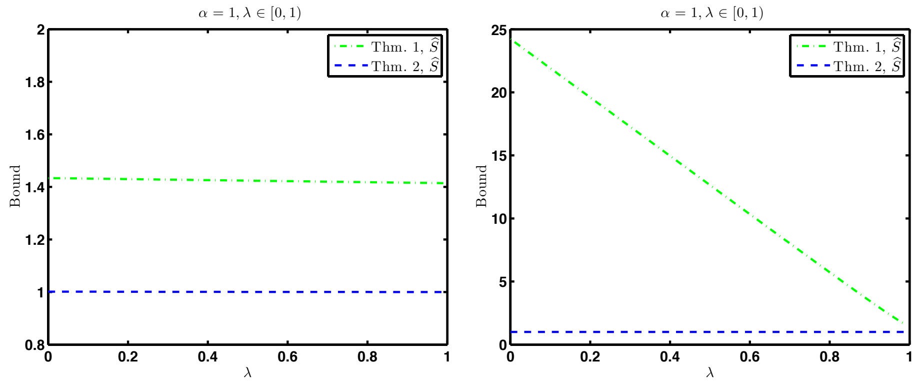
Average Cost and Stochastic Shortest Path (SSP) Problems. In the average cost case (similarly for SSP), x^* is the differential cost or bias vector and it is orthogonal to e. Let us assume that S is orthogonal to e, to simplify



(a) Standard bounds vs. Theorem 1

(b) Standard bounds vs. Theorems 1 & 2 [detail of lower portion of (a)]

Fig. 3. Comparison of error bounds for discounted problems. The setup is the same as that for Figure 2, except that the projection norm is the standard Euclidean norm. The Markov chain has a "noisy" block structure. The subspace S is chosen randomly.



(a) Theorem 1 vs. Theorem 2

(b) Theorem 1 vs. Theorem 2

Fig. 4. Comparison of error bounds for average cost problems with randomly generated Markov chains. The setup is the same as that for Figure 2. In (b), the Markov chain has a "noisy" block structure, and S is chosen as in Figure 2(c).

the discussion. The error bound corresponding to the bound (3), and given by Tsitsiklis and Van Roy [11] is

$$\|x^* - \bar{x}\|_\xi \leq \frac{1}{\sqrt{1-\alpha_\lambda^2}} \|x^* - \Pi x^*\|_\xi \, ,$$

where $\alpha_\lambda < 1$ and $\alpha_\lambda \to 0$ as $\lambda \to 1$. Here, α_λ can be viewed as the modulus of contraction of some mapping that is a damped version of ΠA, while $\alpha_\lambda \to 0$ reflects the fact that the matrix ΠA converges to the zero matrix (as A converges to $e\xi'$) as $\lambda \to 1$. Note that the factor in the bound converges to 1, as $\lambda \to 1$. This bound is qualitative, as usually the value of α_λ is unknown.

Figure 4 shows the bounds of Theorems 1 and 2. Notice that as $\lambda \to 1$, the bound of Theorem 1 converges to $\sqrt{2}$ instead of 1. This is due to the over-relaxation in the analysis for the case where ΠA is near zero, as remarked in Section 2.2. Notice also in Figure 4(b) that the bound of Theorem 1 is affected by the relation of S to the eigenspace of A associated with eigenvalues that are close to 1, similar to the discounted case. By contrast, the bound of Theorem 2 performs well.

3.2 Large General Systems of Linear Equations

For solving large general systems of linear equations using the projected equation approach [3], the bound of Theorem 2 can be computed in a straightforward way (except in the case of TD(λ) with $\lambda > 0$), as shown in Examples 1 and 2. Theorem 2 is not only much sharper than Theorem 1 for this case, but also more convenient, because it does not require the knowledge of $\|A\|_\xi$. Note that we can write linear equations of the form $Lx = q$ as $x = Ax + b$, with $A = I + cL$ and $b = -cq$ for any scalar c, and we can choose c to optimize the corresponding error bound.

4 Discussion

We have considered the projected equation approximation approach, and we have presented new data-dependent computable error bounds that hold for both contraction and non-contraction mappings. Their applicability for non-contraction mappings is not only useful for approximating solutions of general linear equations, but is also useful in the context of MDP for designing exploration mechanisms. Furthermore, in the context of MDP, these bounds can be used in performance bounds for approximate policy iteration, such as [6].

One potential use of our bounds is to suggest changes in the projected equation in order to reduce the amplification ratio. For example, extensive computational experience with TD(λ) methods suggests that the simulation noise tends to increase as λ increases, so there is motivation to use small values of λ as long as the amplification ratio is close to 1. Unfortunately, the bounds (2), (3) are too conservative to provide useful information about the amplification ratio, and our bounds can provide quantitative guidance as well as valuable insight in this regard. Furthermore, our bounds can be similarly used in the general non-contraction context, in conjunction with simulation-based TD(λ)-like algorithms that have been developed in our recent paper [3]. There may be other potential uses of our bounds, for example in suggesting changes to the choice of approximation subspace, thereby affecting both the baseline error and the amplification ratio, but this is a subject for future research.

Acknowledgment. Huizhen Yu is supported in part by Academy of Finland grant 118653 (ALGODAN) and by the IST Programme of the European Community, PASCAL Network of Excellence, IST-2002-506778.

References

1. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 3rd edn., vol. II. Athena Scientific, Belmont (2007)
2. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
3. Bertsekas, D.P., Yu, H.: Projected equation methods for approximate solution of large linear systems. J. Computational and Applied Mathematics (to appear, 2008)
4. Boyan, J.A.: Least-squares temporal difference learning. In: Proc. of the 16th Int. Conf. Machine Learning (1999)
5. Konda, V.R.: Actor-Critic Algorithms. Ph.D thesis. MIT, Cambridge (2002)
6. Munos, R.: Error bounds for approximate policy iteration. In: Proc. The 20th Int. Conf. Machine Learning (2003)
7. Nedić, A., Bertsekas, D.P.: Least squares policy evaluation algorithms with linear function approximation. Discrete Event Dyn. Syst. 13, 79–110 (2003)
8. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning 3, 9–44 (1988)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning. MIT Press, Cambridge (1998)
10. Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal-difference learning with function approximation. IEEE Trans. Automat. Contr. 42(5), 674–690 (1997)
11. Tsitsiklis, J.N., Van Roy, B.: Average cost temporal-difference learning. Automatica 35(11), 1799–1808 (1999)
12. Yu, H., Bertsekas, D.P.: New error bounds for approximations from projected linear equations. Technical Report C-2008-43, University of Helsinki (2008)

Markov Decision Processes with Arbitrary Reward Processes

Jia Yuan Yu[1], Shie Mannor[1], and Nahum Shimkin[2]

[1] McGill University
jia.yu@mcgill.ca, shie.mannor@mcgill.ca
[2] Technion
shimkin@ee.technion.ac.il

Abstract. We consider a control problem where the decision maker interacts with a standard Markov decision process with the exception that the reward functions vary arbitrarily over time. We extend the notion of Hannan consistency to this setting, showing that, in hindsight, the agent can perform almost as well as every deterministic policy. We present efficient online algorithms in the spirit of reinforcement learning that ensure that the agent's performance loss, or regret, vanishes over time, provided that the environment is oblivious to the agent's actions. However, counterexamples indicate that the regret does not vanish if the environment is not oblivious.

1 Introduction

We consider an agent, or decision-maker, interacting with a dynamic environment, whose state evolves as a Markov decision process and whose rewards change arbitrarily from step to step. The arbitrary reward model encompasses non-stationary or unpredictable environments, and opponents that cannot be modeled. Many real-world control systems and complex decision problems can be naturally modeled as Markovian processes in terms of dynamics, but not in terms of rewards. Unless the reward process has well defined characteristics (*e.g.*, [10]), the classical Markov decision processes (MDP's) model cannot be used. Yet, in some of these cases, we can observe, fully or partially, the reward process. As a motivation, let us consider the following example.

Example 1 (Multi-Armed Bandit with Delay). Consider a multi-armed bandit problem with k arms and a special restriction (see Figure 1). When choosing an arm for the following time step, the agent may repeat the current arm or switch. If it chooses to switch to a different arm, it must go through an intermediate idle state before completing the switch. We have one state for each of the k arms, each with two actions ("repeat" and "switch"), and one "idle" state with k actions leading to the corresponding arms. Not only does an MDP naturally model this system, it can further capture the costs to transition from one arm to another. We will show that, under reasonable assumptions, it is possible to achieve asymptotically a reward as high as that of the best arm, regardless of extra delays and transition costs.

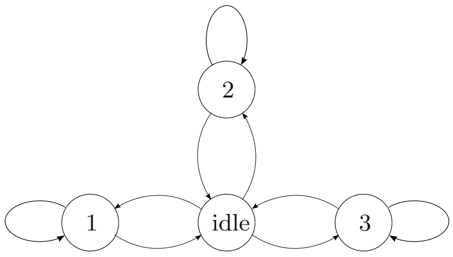


Fig. 1. Example 1: Multi-armed bandit with three arms and idle state

In this work, we make almost no assumption on the reward functions. Within this setting, a reasonable objective is to do as well as some set of alternative policies in hindsight—namely, to *minimize regret*. The concept of regret compares, in retrospect[1], the actual reward cumulated by an agent employing an online learning policy to the highest reward attainable by a set of alternative policies (*e.g.*, the set of deterministic policies). Note that the set of deterministic policies, although rich, pales in comparison with the set of all behavioral policies. This essentially offsets the fact that the agent does not have the benefit of foresight, *i.e.*, it has absolutely no prior knowledge about the rewards. To minimize regret is to achieve zero regret after averaging over time and as the time horizon increases to infinity.

This work extends the objective of regret minimization (surveyed in Section 3) to a dynamic environment modeled as a Markov decision process with an arbitrary sequence of reward functions. We employ an approach where the agent follows a fixed policy over each interval of a partitioned time horizon. We present solutions with the following features: computational efficient, capable to estimate partially observed reward functions, and able to track infrequent changes in the best deterministic policy. We provide performance guarantees in the form of asymptotic bounds on the regret in terms of the length of the time horizon. Dependencies on the sizes of the state and action spaces, and mixing times, are not the focus of this paper.

We begin by describing the setting (Section 2) and situating our work within the literature in Section 3. In Section 4, we present an algorithm that ensure that the agent's expected regret over T steps vanishes almost as[2] $O(T^{-1/4})$ if the environment is oblivious to the agent's state and action history. Furthermore, the regret vanishes with probability 1 as long as the agent does not change its policy too often. Among further extensions, we present modifications of our basic algorithm to reduce the computational cost with approximations, to estimate the average reward process when observation is limited to the trajectory (*i.e.*, the

[1] By "retrospect" or "hindsight," we mean "given complete knowledge of the reward functions picked by the environment."

[2] We adopt the standard order notations. For instance, given sequences f_n and g_n, we write $f_n = O(g_n)$ if and only if there exist N and $\beta > 0$ such that $|f_n| \leq \beta\,|g_n|$ for all $n > N$.

agent only observes rewards in the states visited), and to track the optimal policy through infrequent changes. In Section 5, we show that approximation algorithms for dynamic programming (*e.g.*, Q-learning type algorithms) can be used to minimize the regret when an exact solution is too demanding. Section 6 shows that the agent does *not* have to observe the entire reward functions, but only the rewards at state-action pairs that are visited. In Section 7, we minimize regret with respect to a subset of non-stationary policies—dynamic policies where the number of policy changes from step to step is limited.

2 Problem Definition

We consider an agent in a dynamic environment with two components: a controlled Markov process and an arbitrarily time-varying reward process. The reward process is assumed to behave in a *non-stochastic* and possibly arbitrary manner. It can be thought of as driven by an opponent or Nature. The Markov chain component is a standard *Markov decision process* (MDP) that is defined by a triple (S, A, P) where S is the (finite) set of states of the MDP, A is the (finite) set of actions available to the agent (assumed identical in all states, without loss of generality), and P is the transition probability—that is, $P(s' \mid s, a)$ is the probability that the next state is s' if the current state is s and action a is taken.

The discrete steps are indexed $t = 0, 1, \ldots$ We assume throughout the paper that the initial state at step 0 is fixed and denoted s_0, unless otherwise specified. At the t-th step, the following happen:

1. The opponent chooses a reward function $r_t : S \times A \to [0, 1]$;
2. the state s_t is revealed;
3. the agent chooses an action a_t;
4. the entire reward function r_t is revealed; the agent receives reward $r_t(s_t, a_t)$;
5. the next state $\mathbf{s}_{t+1}$ is determined stochastically according to the transition probability $P(s_{t+1} \mid s_t, a_t)$.

Remark 1 (Notation). We associate random variables with a bold typeface (*e.g.*, $\mathbf{s}_t$), and their realizations with a normal typeface (*e.g.*, s_t).

In general, the *opponent* (a term that we use synonymously with *environment*) determines a sequence of functions $r_0, r_1, \ldots$, where r_t may be picked on the basis of the past state-action history, *i.e.*, on $(s_0, a_0, \ldots, s_{t-1}, a_{t-1})$. In most of the following development, we consider oblivious opponents that pick the reward functions $r_1, r_2, \ldots$ independently of the past state-action history. This is made exact in the following section.

We are interested in agents that learn online. When choosing action a_t at step t, we assume that the agent knows the current state s_t, as well as the transition probability P and the past reward functions $r_0, \ldots, r_{t-1}$ over all state-action pairs. A *policy* is a mapping from the reward history $(r_0, \ldots, r_{t-1})$ and transition history $(s_0, a_0, s_1, \ldots, s_t)$ to a mixed action in the simplex $\Delta(A)$[3]. A *stationary*

[3] $\Delta(A)$ denotes the set of all probability vectors over A.

policy is a policy $\mu : S \to \Delta(A)$ that depends solely on the current state s_t—and not on the history of the rewards or the transitions. A stationary deterministic policy, or *deterministic* policy for short, is a mapping $\mu : S \to A$ from the current state to an action. We shall denote by $\mathbf{a}_t$ the agent's (random) action at step t, chosen according to its policy.

2.1 Regret

The goal of the agent is to maximize its average cumulative reward over a long time horizon of T steps, which is not known a priori: $\frac{1}{T}\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t)$. We shall focus on policies that minimize regret, which measures in retrospect how worse off the agent is compared to a set of alternative policies. Regret arises from the lack of prior knowledge on the sequence of reward functions picked by the opponent. We present a notion of regret that essentially compares the actual reward obtained by the agent with a hypothetical reward. This hypothetical reward is the expected reward an agent that possesses prior knowledge concerning the reward function sequence could have obtained. We restrict our basis of comparison to the *set of deterministic policies* $\{\mu : S \to A\}$, which we denote by Σ. Our concept of regret collapses to the classical notion of regret studied in online learning (cf. [8]).

A natural hypothetical reward is the expected reward attainable by the best deterministic policy given full knowledge of the sequence of reward functions r_t. Hence, we have the following definition, which extends the concept of "modified regret" introduced in [13].

Definition 1 (Regret). *The regret is*

$$\sup_{\mu \in \Sigma} \mathbb{E}\left[\sum_{t=0}^{T-1} r_t(\tilde{\mathbf{s}}_t, \mu(\tilde{\mathbf{s}}_t))\right] - \sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t), \tag{1}$$

where the expectation $\mathbb{E}$ is over the sequence $(\tilde{\mathbf{s}}_t, \mu(\tilde{\mathbf{s}}_t))$. This is a random *quantity that depends on the agent's random trajectory $(\mathbf{s}_t, \mathbf{a}_t)$. It is implicitly understood that both sequences $\tilde{\mathbf{s}}_t$ and $\mathbf{s}_t$ start at the initial state s_0 and follow the transition kernel P.*

Observe that this notion of regret remain the same whether Σ is the set of deterministic policies, or the set of stationary policies—as considered by [16] and [11].

3 Related Works

The problem of minimizing the regret can be traced back to the seminal work of [13]. In the context of repeated matrix games where the sequence of opponent actions is arbitrary, Hannan proposed a policy that adapts to the opponent's past actions in an incremental fashion, yet asymptotically performs as well as the best fixed policy against the same sequence of opponent actions. In fact, the difference in cumulated payoff between these policies is only of the order of the square root of the number of repetitions of the game, regardless of whether the opponent is oblivious or not. The proposed policy optimizes the payoff against the empirical distributions of the opponent's actions so far perturbed by vanishingly small noise. The reader can find a survey of methods for achieving Hannan consistency in [8].

Optimal control in MDP's with unknown, but stationary, reward processes can be solved using reinforcement learning, *e.g.*, model-based and Q-learning algorithms [4]. However, fairly limited work has been done on regret minimization for Markovian environments. One model of interest is a stochastic game [18] where rewards and transitions depend on the actions of an arbitrary opponent, as in [16]. In contrast to an ordinary stochastic game, the opponent is not necessarily rational or self-optimizing. The emphasis is providing the agent with policies that perform well against *every* possible opponent. An equilibrium solution to a zero-sum stochastic game, such as one produced by the R-max algorithm of [7], may well be too conservative when the opponent is not adversarial. In such a setting, the agent's goal is to exploit the non-adversarial characteristic of the opponent. Using approachability theory, Mannor and Shimkin [16] show that regret minimization is possible, but only with respect to a relaxed objective. Notably, single-controller games, where the opponent alone controls the transition probabilities, can be treated as a sequence of interleaved repeated games, and regret can be minimized by separately minimizing the regret in each repeated game. Our model corresponds to a stochastic game where an arbitrary opponent picks the reward functions, but does not affect state transitions. We give algorithmic solutions with clear computational complexity guarantees. Merhav *et al.* [17] consider sequential decision problems where the loss functions have memory, which correspond to MDP's where every state is reachable from every other via a single action. They present an algorithm using piecewise-constant policies and provide regret-minimizing guarantees similar to ours.

The problem that we consider resembles that of Even-Dar *et al.* [11]. Although the motivation is the same, our model differs significantly. In our model, the instantaneous reward to the agent is a *random variable* $r_t(\mathbf{s}_t, \mathbf{a}_t)$ and, in turn, the regret is defined in terms of a random state-action trajectory.

4 Follow the Perturbed Leader

In this section, we present and analyze an algorithm that minimizes regret. To preclude singular examples, we shall first introduce an uniform ergodicity assumption on the MDP structure and an obliviousness assumption on the opponent. The algorithm and its analysis will follow.

Assumption 1 (Uniform Ergodicity). The induced Markov chain is uniformly ergodic over the set of deterministic policies[4]. This guarantees that there

[4] The ergodic assumption is quite weak as it only requires that all recurrent states in the Markov chain communicate and be aperiodic.

exists a unique stationary distribution $\pi(\mu)$ for each policy μ. Moreover, there exists (cf. [5]) a finite mixing time $\gamma \geq 0$; *i.e.*, there exists a finite $\gamma \geq 0$ such that for every policy $\mu \in \Sigma$, every initial state s_0, and $t \geq 0$, we have

$$\|d_t(\mu; s_0) - \pi(\mu)\|_1 \leq 2e^{1-t/\gamma}.$$

4.1 Oblivious Opponent

In this section, we present models for the environment against which online learning can not guarantee vanishing regret.

Example 2 (Non-oblivious Opponent). Let the states be $S = \{1, 2, 3\}$ as in Figure 2. The agent has two actions $A = \{\alpha_1, \alpha_2\}$ to choose from: whether to move from state 1 to state 2 (action α_2) or from state 1 to state 3 (action α_3). There is a small probability of staying in state 1 regardless of the action, thus making the MDP aperiodic. From state 2 or 3, the agent moves back to state 1 deterministically.

The non-oblivious opponent assigns 0 reward to state 1 at all stages. It gives a reward of 1 to the state 2 if the agent took the action leading to state 3 at the previous time step; otherwise, it gives 0 reward to state 2. Similarly, the opponent gives a reward of 1 to the state 3 if the agent took the action leading to state 2, and a reward of 0 otherwise. Consequently, for every policy, the attained reward is $\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t) = 0$, whereas we have either $\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \alpha_2) \geq 1/2 - p$ or $\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \alpha_3) \geq 1/2 - p$. As a result, the average regret is always positive and bounded away from 0. Hence, for this aperiodic and recurrent MDP, zero-regret policies do not exist. This example is stronger than the counterexample presented in [16], where the non-vanishing regret is attributed to lack of observation of the reward.

In light of this counterexample, we restrict our attention to problems where the opponent is oblivious; in other words, when the opponent determines the entire sequence of reward functions ahead of time, but reveals them one at a time. There are two justifications for this approach. First, from a modeling perspective, the agent may interact with other agents that are truly oblivious, irrational, or have

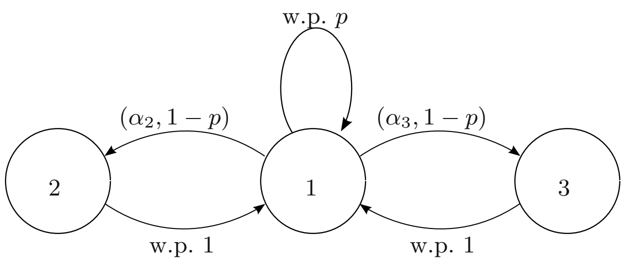


Fig. 2. Example 2. Taking action α_2 in state 1 leads to state 2 with probability $1-p$. Moreover, taking any action in state 1 leads back to state 1 with probability p.

an unspecified or varying objective. This renders their behavior "unpredictable" and seemingly arbitrary. Second, in the presence of many agents, a single agent has little effect on the overall outcome (*e.g.*, price of commodities, traffic in networks) due to the effect of large numbers [2]. Moreover, as Example 2 shows, the regret cannot be made asymptotically small when the opponent is not oblivious. Formally, we state the non-obliviousness assumption as follows.

Assumption 2 (Oblivious Opponent). The reward functions r_t, for $t = 0, \ldots, T-1$, are random variables on the null σ-algebra. Hence, for every random variable $\mathbf{X}_t$ measurable by the σ-algebra generated by $(\mathbf{s}_0, \mathbf{a}_0, r_0, \ldots, \mathbf{s}_t, \mathbf{a}_t)$ satisfies the following:

$$\mathbb{E}\big[r_t(s,a)\mathbf{X}_t\big] = r_t(s,a)\mathbb{E}\big[\mathbf{X}_t\big], \quad \text{for all } (s,a) \in S \times A.$$

4.2 Algorithm

We present an algorithm based on a concept due to [13], that of following the best action so far subject to some random perturbation that vanishes with time. The algorithm works in phases. We partition the steps $\{0, \ldots, T-1\}$ into M phases (*i.e.*, intervals of consecutive steps[5]), denoted by $\tau_0, \ldots, \tau_{M-1}$. The phases are long enough that the state-action distribution approaches stationarity, and such that the number of phases M becomes sublinear in T—this will be made precise in the results below. The phases are nonetheless short enough so that the agent adapts fast enough to changes in the reward functions. As a convention, we let the index t denote a step, whereas the index m refers to phase τ_m. Moreover, we write $\tau_0 \ldots \tau_m$ to denote the union of phases $\tau_0 \cup \ldots \cup \tau_m$, and $|\tau_0 \ldots \tau_m|$ to denote its length.

For ease of notation, we write the average reward over one or more phases as

$$\widehat{r}_{\tau_m}(s,a) \triangleq \frac{1}{|\tau_m|}\sum_{t\in\tau_m} r_t(s,a), \quad \widehat{r}_{\tau_0\ldots\tau_m}(s,a) \triangleq \frac{1}{|\tau_0\ldots\tau_m|}\sum_{t\in\tau_0\ldots\tau_m} r_t(s,a),$$

for all $(s,a) \in S \times A$. The algorithm takes as input the step index $t \in \tau_m$, state s_t, and average reward function $\widehat{r}_{\tau_0\ldots\tau_{m-1}}$. It outputs a random action $\mathbf{a}_t(s_t)$ distributed according to a mixed policy σ_m. For the purpose of randomization, the algorithm samples a sequence $\mathbf{n}_1, \ldots, \mathbf{n}_{M-1}$ of independent random variables[6] in $\mathbb{R}^{|S\times A|}$.

Algorithm 1 (Lazy FPL).

1. (*Initialize.*) For $t \in \tau_0$, choose the action $\mathbf{a}_t(s_t)$ according to an arbitrary deterministic policy $\mu : S \to A$.
2. (*Update.*) At the start of phase τ_m, for $m = 1, 2, \ldots$, solve the following linear program for (λ_m, h_m):

[5] The partition is constructed such that the order between steps within each phase is preserved.

[6] Their distributions will be specified later.

$$\min_{\lambda \in \mathbb{R}, h \in \mathbb{R}^{|S|}} \quad \lambda \tag{2}$$

$$\text{subject to:} \quad \lambda + h(s) \geq \widehat{r}_{\tau_0 \ldots \tau_{m-1}}(s,a) + \sum_{s' \in S} P(s' \mid s,a) h(s'), \quad (s,a) \in S \times A,$$

$$h(s^+) = 0, \quad \text{for some fixed } s^+ \in S.$$

3. (*Follow the perturbed leader.*) For $t \in \tau_m$, for $m = 1, 2, \ldots$, choose the action

$$\mathbf{a}_t(s_t) = \arg\max_{a \in A} \left\{ \widehat{r}_{\tau_0 \ldots \tau_{m-1}}(s_t, a) + \mathbf{n}_m(s_t, a) + \mathbb{E}_{\mathbf{s}_{t+1} | s_t, a} \left[h_m(\mathbf{s}_{t+1}) \right] \right\}, \tag{3}$$

where the element of A with the lowest index is taken if the max is not unique, and the expectation is only taken with respect to the transition probabilities from the current state s_t to the next one $\mathbf{s}_{t+1}$, *i.e.*,

$$\mathbb{E}_{\mathbf{s}_{t+1} | s_t, a} \left[h_m(\mathbf{s}_{t+1}) \right] = \sum_{s' \in S} P(s' \mid s_t, a) h_m(s').$$

Observe that the linear program (2) solves MDP's with the average-reward as the objective [3]. The Lazy FPL algorithm perturbs the average reward function $\widehat{r}_{\tau_0 \ldots \tau_{m-1}}$ with a random variable $\mathbf{n}_m$, so that $\mathbf{a}_t$ is random as well. The algorithm samples the random variable $\mathbf{n}_m$ only once at the start of every phase. The realization of $\mathbf{n}_m$ is used throughout the phase τ_m. Therefore, for a fixed state s, the actions $\mathbf{a}_t(s)$, for $t \in \tau_m$, are identical. In effect, there exists a deterministic policy that determines the actions of each phase. We shall denote by $\sigma_m : S \to \Delta(A)$ the mixed policy corresponding to the distribution of $\mathbf{a}_t$ for $t \in \tau_m$.

Introducing randomness through perturbations guarantees that the policies in consecutive phases do not change too abruptly. We thus avoid behavior reminiscent of the proverbial Buridan's ass. This is similar to other regret minimization algorithms (*e.g.*, [13,15]) and smooth fictitious play [12]. The motivation of increasing phase lengths is twofold. Firstly, using a fixed policy over long phases is computationally efficient. Secondly, in addition to vanishing expected regret, we show vanishing regret with probability 1, provided that the agent does not change its policy too often. One intuition is that, on the one hand, our bases for comparison are the steady-state rewards of deterministic policies; on the other hand, taking long phases ensures that the agent's accumulated reward in each phase approaches the steady-state reward of the corresponding policy. Finally, observe that prior knowledge of the time horizon T is not necessary to run the Lazy FPL algorithm. The only prerequisite is a pre-established scheme to partition every time interval into phases.

4.3 Results

In this section, we show that if we follow the Lazy FPL algorithm, then the time-average regret converges almost-surely to zero, *i.e.*, Hannan consistency. For the sake of space, the proofs are confined to [19].

The following main result shows that increasing phase lengths yields not only an efficient implementation, but also guarantees a stronger type of convergence for the average regret.

Theorem 1 (Hannan Consistency of Lazy FPL). *Suppose that Assumptions 1 and 2 hold. Further, suppose that the random variables* $\mathbf{n}_m(s,a)$, *for* $m = 1, 2, \ldots$ *and* $(s,a) \in S \times A$, *are independent and uniformly distributed*[7] *over the support* $[-1/\zeta_m, 1/\zeta_m]$, *where* $\zeta_m \triangleq \sqrt{|\tau_0 \ldots \tau_m|}$. *Let the time horizon* $0, \ldots, T-1$ *be partitioned into phases* $\tau_0, \ldots, \tau_{M-1}$ *such that there exists an* $\epsilon \in (0, 1/3)$ *for which* $|\tau_m| = \lceil m^{1/3-\epsilon} \rceil$, *for* $m = 0, \ldots, M-1$. *Then, the average regret of the Lazy FPL algorithm vanishes almost surely,* i.e.,

$$\limsup_{T\to\infty} \left\{ \sup_{\mu\in\Sigma} \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} r_t(\tilde{\mathbf{s}}_t, \mu(\tilde{\mathbf{s}}_t))\right] - \frac{1}{T}\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t) \right\} \le 0, \quad w.p.\ 1.$$

Remark 2. Theorem 1 makes no assumption about the sequence of reward functions except for boundedness and obliviousness.

Remark 3. Observe that the partition of Theorem 1 can be constructed incrementally over time, without prior knowledge of the time horizon T. Having a slowly increasing phase length suffices for obtaining convergence.

The proof of Theorem 1 (cf. [19]) proceeds as follows. The oblivious opponent assumption makes stationary policies as good as any other within long phases. The ergodicity assumption allows us to concentrate on the stationary distributions of the baseline policies, as well as the policies of the sequence of phases. The perturbation noise endows a continuity between policies of consecutive phases, yet vanishing quickly enough as not to severely affect the optimality of the solution computed in the Update step of the Lazy FPL algorithm. Next, a careful choice of the lengths of the phases ensures that the agent cannot much improve its total reward had it adopted the best stationary policy against the observed reward functions. Finally, a probabilistic bound on the regret is derived using a modified version of Azuma's Inequality.

Proposition 1 (Expected Regret Bound). *Suppose that the assumptions of Theorem 1 hold. In particular, suppose that there exists an* $\epsilon \in (0, 1/3)$ *such that* $|\tau_m| = \lceil m^{1/3-\epsilon} \rceil$, *for* $m = 0, \ldots, M-1$. *Then, the time-averaged expected regret of the Lazy FPL algorithm satisfies the following bound:*

$$\begin{aligned} &\sup_{\mu\in\Sigma} \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} r_t(\tilde{\mathbf{s}}_t, \mu(\tilde{\mathbf{s}}_t))\right] - \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t)\right] \\ &\le \frac{4}{3}\left(2e\gamma + 2|A| + 4e + 1 + 2(|S|+3)|A|^2 \gamma \log(T)\right) T^{-1/4+\epsilon}. \end{aligned} \tag{4}$$

[7] The random variable $\mathbf{n}_m(s,a)$ has probability density function

$$f_{\mathbf{n}_m(s,a)}(z) = \begin{cases} \zeta_m/2, & \text{if } z \in [-1/\zeta_m, 1/\zeta_m], \\ 0, & \text{otherwise.} \end{cases}$$

5 Approximate Algorithms

In many cases, computing the optimal policy σ_m at each phase τ_m in the Lazy FPL algorithm, might be intractable. We may instead compute an approximation ρ_m to σ_m. The policy ρ_m is still computed once every phase, *i.e.*, once every $|\tau_m|$ steps, but using a computationally efficient method. Our approximation proceeds by approximating the optimal Q-function. Recall that a Q-function is a function $Q : S \times A \to \mathbb{R}$ that represents the relative advantage of using a particular action at a particular state. Let h_m denote a vector that is part of the optimal solution to the linear program (2) at the start of phase τ_m. The corresponding optimal Q-function is therefore defined as

$$Q^*_m(s,a) = \widehat{r}_{\tau_0\ldots\tau_{m-1}}(s,a) + \sum_{s' \in S} P(s' \mid s,a) h_m(s').$$

In the following algorithm, we use Q-learning [4, Chapter 7] to compute an approximation ρ_m to the optimal policy σ_m of the Lazy FPL algorithm. In essence, Q-learning is employed as a method of solving the linear program of the Lazy FPL algorithm. It is well known that Q-learning is an iterative simulation-based method (reminiscent of value iteration) that does not need to keep track of the transition probabilities. Let Q_t denote the sequence of Q-functions over $S \times A$, and $Q_{\tau_0\ldots\tau_{m-1}}$ denote the Q-function obtained at the last step of phase τ_{m-1}. During every step t of phase τ_m, we choose our action to maximize the Q-function $Q_{\tau_0\ldots\tau_{m-1}}$ obtained over the previous phases, perturbed by a random noise term $\mathbf{n}_m$; simultaneously, we update the Q-function at every step.

Algorithm 2 (Q-FPL).

1. (*Initialize.*) For $t \in \tau_0$, set $Q_t = 0$ and choose action $\mathbf{a}_t(s_t)$ according to an arbitrary deterministic policy $\mu : S \to A$.
2. (*Update.*) At every step $t \in \tau_m$, for $m = 1, 2, \ldots$, set $\kappa_m = 1/\sqrt{m}$ and update Q_t iteratively as follows:

$$\begin{aligned} Q_t(s_{t-1}, a_{t-1}) &= (1-\kappa_m) Q_{t-1}(s_{t-1}, a_{t-1}) \\ &+ \kappa_m \Big(\widehat{r}_{\tau_0\ldots\tau_{m-1}}(s_{t-1}, a_{t-1}) + \max_{a \in A} Q_{t-1}(s_t, a) - Q_{t-1}(s', a') \Big), \end{aligned} \tag{5}$$

   where s' and a' are fixed, and the term $Q_{t-1}(s', a')$ is for the purpose of normalization.
3. (*Perturb.*) At every step $t \in \tau_m$, for $m = 1, 2, \ldots$, choose action

$$\mathbf{a}_t(s_t) = \arg\max_{a \in A} \left\{ Q_{\tau_0\ldots\tau_{m-1}}(s_t, a) + \mathbf{n}_m(s_t, a) \right\},$$

   where $\mathbf{n}_m$ conforms to the assumptions of Theorem 1.

Remark 4. As for the Lazy FPL algorithm, the reward function $\widehat{r}_{\tau_0\ldots\tau_{m-1}}$ is fixed throughout phase τ_m.

Let $Q^*_{\tau_0\cdots\tau_{m-1}}$ denote the optimal Q-function in phase τ_m with the fixed reward function $\widehat{r}_{\tau_0\ldots\tau_{m-1}}$. By [6, Theorem 2.4], within each phase of our MDP—where the reward function is fixed, for every $\beta > 0$ and $\gamma > 0$, and phases long enough, we have $\Pr\big(\|Q_t - Q^*_{\tau_0\ldots\tau_{m-1}}\|_1 > \beta\big) < \gamma$, as required[8]. The sequence κ_m is selected such that it tends to zero at a sub-linear rate (see the discussion in Section 4.3 of [6]). We obtain the following corollary by an argument similar to Theorem 1.

Corollary 1. *Suppose that the assumptions of Theorem 1 hold. Then, the average regret of the Q-FPL algorithm vanishes almost surely.*

Other algorithms, especially some actor-critic algorithms that are equivalent to Q-learning [9], may be used as well.

Remark 5 (Computational Load). The Q-FPL algorithm has a fixed computational load per step. This complexity is less demanding than that of [11], although the latter is also fixed per step. In comparison, the Lazy FPL algorithm requires solving an MDP at the beginning of every phase, but it has the advantage of diminishing the per-step complexity.

6 Observing Rewards Only on Trajectory

We present a modification of the Lazy FPL algorithm in the spirit of [1] to deal with instances where the reward functions are partially observed. More precisely, we consider the case where the agent only observes the value of the reward function sequence on its state-action history (or trajectory). Consequently, we restrict the space of the agent's policies to those that map observed rewards $r_0(s_0, a_0), \ldots, r_{t-1}(s_{t-1}, a_{t-1})$ and the current state s_t to a mixed action. Our approach is to construct an unbiased estimate of $\widehat{r}_{\tau_0\ldots\tau_{m-1}}$ for all m.

To this end, we construct a *random* reward function at every step t. The length of the phase τ_0 and the policy adopted therein are such that $P\big((\mathbf{s}_t, \mathbf{a}_t) = (s, a) \mid s_0\big) > 0$ for $t \geq |\tau_0|$ and $(s, a) \in S \times A$. Next, for all $t \geq |\tau_0|$ and $(s, a) \in S \times A$, let

$$\mathbf{z}_t(s, a) = \begin{cases} \frac{r_t(s,a)}{P\big((\mathbf{s}_t,\mathbf{a}_t)=(s,a) \mid s_0\big)}, & \text{if } (\mathbf{s}_t, \mathbf{a}_t) = (s, a), \\ 0, & \text{otherwise.} \end{cases}$$

Observe that only the value of r_t at the traversed state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ is required to evaluate $\mathbf{z}_t$. The probability $P\big((\mathbf{s}_t, \mathbf{a}_t) = (s, a) \mid s_0\big)$ is readily computed recursively using the transition probabilities associated with the policy followed at step $t-1$. From the sequence $\mathbf{z}_j$, we construct $\widehat{\mathbf{z}}_t \triangleq \frac{1}{t}\sum_{j=0}^{t-1} \mathbf{z}_j$ as an estimate of $\widehat{r}_t = (1/t)\sum_{j=0}^{t-1} r_j$. In conformance with our notation, $\widehat{\mathbf{z}}_{\tau_0\ldots\tau_{m-1}}$ denotes $\widehat{\mathbf{z}}_t$, where t is the first step of phase τ_m.

[8] To be accurate, for the off-policy Q-function evaluation in Step 2 of the Q-FPL algorithm to converge to $Q^*_{\tau_0\ldots\tau_{m-1}}$, we must ensure that the policy induced by Step 3 performs sufficient exploration.

Algorithm 3 (Exploratory FPL).

1. (*Initialize.*) Let the length of phase τ_0 be long enough that $P\big((\mathbf{s}_t, \mathbf{a}_t) = (s, a) \mid s_0\big) > 0$ for $t \geq |\tau_0|$ and $(s, a) \in S \times A$. For $t \in \tau_0$, choose action $\mathbf{a}_t$ uniformly at random over A.
2. (*Estimate.*) At every step $t = 1, 2, \ldots$, compute the estimate $\widehat{\mathbf{z}}_t$ recursively.
3. (*Sample.*) At the start of phase τ_m, for $m = 1, 2, \ldots$, sample an independent Bernoulli random variable $\mathbf{x}_m$ that takes value 1 with probability ϕ_m.
4. (*Lazy FPL.*) If $\mathbf{x}_m = 0$, by substituting $\widehat{\mathbf{z}}_{\tau_0 \ldots \tau_{m-1}}$ for $\widehat{r}_{\tau_0 \ldots \tau_{m-1}}$, solve the linear program (2) and follow the policy of Equation (3) throughout phase τ_m.
5. (*Explore.*) If $\mathbf{x}_m = 1$, for $t \in \tau_m$ and $m = 1, 2, \ldots$, choose action $\mathbf{a}_t$ uniformly at random over A.

Corollary 2 (Hannan Consistency of Exploratory FPL). *Suppose that the assumptions of Theorem 1 hold. Further, suppose that the agent follows the Exploratory FPL algorithm with a sequence ϕ_m ensuring infinitely many exploration phases, and such that*

$$\sum_{m=0}^{M-1} |\tau_m| \, \phi_m = O(M), \quad \text{and} \quad \phi_m > 0, \quad \text{for } m = 0, \ldots, M-1. \tag{6}$$

Then, the average regret of the Exploratory FPL algorithm vanishes almost surely.

Remark 6. Corollary 2 guarantees that the Exploratory FPL algorithm minimizes regret in the generalized multi-arm bandit problem of Example 1.

7 Regret against Dynamic Policies

In this section, we consider a more general notion of regret that encompasses some dynamic policies. Consider a sequence of policies $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{T-1})$, where every element μ_j of this sequence is a deterministic policy $\mu_j : S \to A$. Let the number of switches in this sequence of policies be $K(\boldsymbol{\mu}) = \sum_{j=1}^{T-1} 1_{[\mu_{j-1} \neq \mu_j]}$. Let K_0 be a fixed integer. A more challenging baseline of comparison for the cumulated reward is

$$\sup_{\substack{(\mu_0, \ldots, \mu_{T-1}): \\ K(\boldsymbol{\mu}) \leq K_0}} \sum_{t=0}^{T-1} \mathbb{E}\big[r_t(\tilde{\mathbf{s}}_t, \mu_t(\tilde{\mathbf{s}}_t))\big], \tag{7}$$

where $(\tilde{\mathbf{s}}_0, \mu_0(\tilde{\mathbf{s}}_0)), \ldots, (\tilde{\mathbf{s}}_{T-1}, \mu_{T-1}(\tilde{\mathbf{s}}_{T-1}))$ denote state-action pairs induced by the sequence of policies $\mu_0, \ldots, \mu_{T-1}$, and the maximum is taken over all possible sequences of policies with at most K_0 switches. If $K_0 = 0$, then Equation (7) reduces to the baseline considered so far (cf. Equation (1)). We present an algorithm that guarantees a reward consistent with the above baseline. This algorithm adapts the Fixed-share algorithm of [14] to the MDP framework.

Algorithm 4 (Tracking FPL).

1. (*Initialize.*) Fix $\alpha \in [0,1]$. For $t \in \tau_0$, choose action $\mathbf{a}_t(s_t)$ according to an arbitrary deterministic policy $\mu : S \to A$.
2. (*Sample.*) At the outset of phase τ_m, for $m = 1, 2, \ldots$, sample a Bernoulli random variable $\mathbf{x}_m$ with $\Pr(\mathbf{x}_m = 1) = \alpha$.
3. (*Fixed-share.*) If $\mathbf{x}_m = 0$, sample a policy $\mathbf{y}_m$ uniformly at random from the set of deterministic policies Σ, then follow the policy $\mathbf{y}_m$ throughout phase τ_m.
4. (*Lazy FPL.*) If $\mathbf{x}_m = 1$, solve the linear program (2) and follow the policy of Equation (3) throughout phase τ_m.

Remark 7. Observe that as before, the algorithm elects a single policy in each phase and follows it throughout. The fixed-share scheme occurs *once in each phase*—at the outset. Observe as well that there exist methods to efficiently construct the uniformly random policy $\mathbf{y}_m$. As in the Fixed-share algorithm of [14], the action at each step is equal to the previous action with probability $1 - \alpha + \alpha/|A|$, and equal to each different action with probability $\alpha/|A|$.

The following theorem guarantees that the regret with respect to the reward achieved by the best sequence of policies with a finite number of switches vanishes asymptotically if the agent employs the Tracking FPL algorithm.

Theorem 2 (Hannan Consistency of Tracking FPL). *Suppose that the assumptions of Theorem 1 hold. Let K_0 be a positive integer. Suppose further that the agent follows the Tracking FPL algorithm with the parameter $\alpha = K_0/(\lceil T/\lceil T^{1/3}\rceil\rceil - 1)$. Then, the average regret with respect to the baseline of Equation (7) vanishes almost surely,* i.e.,

$$\limsup_{T\to\infty} \left\{ \sup_{\substack{(\mu_0,\ldots,\mu_{T-1}):\\ K(\boldsymbol{\mu})\le K_0}} \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} r_t(\tilde{\mathbf{s}}_t, \mu_t(\tilde{\mathbf{s}}_t))\right] - \frac{1}{T}\sum_{t=0}^{T-1} r_t(\mathbf{s}_t, \mathbf{a}_t) \right\} \le 0, \quad \textit{w.p. } 1.$$

Remark 8. Although we only consider the case of a fixed number of switches K_0 and a fixed parameter α, it can be shown, by using doubling trick of [8, Section 3.2], that the result of Theorem 2 holds as long as the number of switches K_0 increases slowly enough in T.

References

1. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The nonstochastic multi-armed bandit problem. SIAM J. Computing 32(1), 48–77 (2002)
2. Aumann, R.J.: Markets with a continuum of traders. Econometrica 32, 39–50 (1964)
3. Bertsekas, D.P.: Dynamic programming and optimal control, 2nd edn., vol. 2. Athena Scientific (2001)

4. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-dynamic programming. Athena Scientific (1996)
5. Bobkov, S.G., Tetali, P.: Modified logarithmic Sobolev inequalities in discrete settings. Journal of Theoretical Probability 19(2), 289–336 (2006)
6. Borkar, V.S., Meyn, S.P.: The O.D.E. method for convergence of stochastic approximation and reinforcement learning. SIAM J. Control and Optimization 38(2), 447–469 (2000)
7. Brafman, R.I., Tennenholtz, M.: R-max—a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3, 213–231 (2003)
8. Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge University Press, Cambridge (2006)
9. Crites, R.H., Barto, A.G.: An actor/critic algorithm that is equivalent to Q-learning. In: Advances in Neural Information Processing Systems, pp. 401–408 (1995)
10. Duffield, N.G., Massey, W.A., Whitt, W.: A nonstationary offered-load model for packet networks. Telecommunication Systems 16(3–4), 271–296 (2001)
11. Even-Dar, E., Kakade, S., Mansour, Y.: Experts in a Markov decision process. In: NIPS, pp. 401–408 (2004)
12. Fudenberg, D., Kreps, D.M.: Learning mixed equilibria. Games and Economic Behavior 5(3), 320–367 (1993)
13. Hannan, J.: Approximation to Bayes risk in repeated play. In: Contributions to the Theory of Games, vol. 3, pp. 97–139. Princeton University Press, Princeton (1957)
14. Herbster, M., Warmuth, M.K.: Tracking the best expert. Machine Learning 32(2), 151–178 (1998)
15. Kalai, A., Vempala, S.: Efficient algorithms for online decision problems. Journal of Computer and System Sciences 71(3), 291–307 (2005)
16. Mannor, S., Shimkin, N.: The empirical Bayes envelope and regret minimization in competitive Markov decision processes. Mathematics of Operations Research 28(2), 327–345 (2003)
17. Merhav, N., Ordentlich, E., Seroussi, G., Weinberger, M.J.: On sequential strategies for loss functions with memory. IEEE Trans. Inf. Theory 48(7), 1947–1958 (2002)
18. Shapley, L.: Stochastic games. PNAS 39(10), 1095–1100 (1953)
19. Yu, J.Y., Mannor, S., Shimkin, N.: Markov decision processes with arbitrarily varying rewards (Preprint, 2008), http://www.cim.mcgill.ca/~jiayuan/mdp.pdf

Author Index