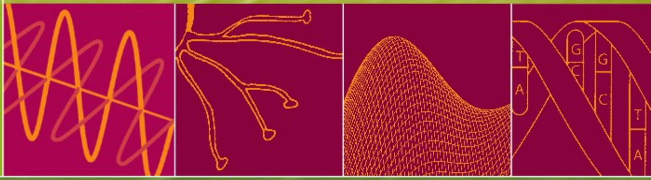
Quantum Computing

Neural Networks

Evolutionary Computing

DNA Computing



NATURAL COMPUTING SERIES

Marco Tomassini

Spatially Structured Evolutionary Algorithms

Artificial Evolution in Space and Time

Springer

Natural Computing Series

Marco Tomassini

Spatially Structured Evolutionary Algorithms

Artificial Evolution in Space and Time

With 91 Figures and 21 Tables

Springer

Marco Tomassini
Information Systems Department (INFORGE)
University of Lausanne
1015 Lausanne
Switzerland
marco.tomassini@unil.ch

Series Editors
G. Rozenberg (Managing Editor)
rozenber@liacs.nl
Th. Bäck, J.N. Kok, H.P. Spaink
Leiden Center for Natural Computing
Leiden University
Niels Bohrweg 1
2333 CA Leiden, The Netherlands

A.E. Eiben
Vrije Universiteit Amsterdam
The Netherlands

Library of Congress Control Number: 2005929365

ACM Computing Classification (1998): F.1-2, G.1.6, G.2.2, I.2.8

ISBN-10 3-540-24193-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-24193-5 Springer Berlin Heidelberg New York

Cover Design: KünkelLopka, Werbeagentur, Heidelberg
Typesetting: by the Author
Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Printed on acid-free paper 45/3142/YL – 5 4 3 2 1 0

To Anne, Frédéric, and Vincent

Preface

The field of evolutionary computation (EC) can no longer be considered an esoteric one. Today, after about thirty years of research, a rich corpus of theory exists and many successful real-life applications are witnessing the usefulness of EC heuristics in many fields. EC is a family of population-based methodologies inspired by the interplay of natural selection and variation. However, both the theory and the applications have tended to focus mainly on mixing populations, also called *panmictic* populations. These are populations in which there is no particular structure: any member of the population is equally likely to "meet" any other member. But Darwin realized long ago that populations may have a spatial structure and that this spatial structure may have an influence on population dynamics. For instance, he remarked how, when they were isolated on islands, some species evolved differently from others that lived in more open environments. If we look around us, we too find that geographical separation factors have helped shape evolution. There are many examples, but one well-documented case is the spread of genomic traits in human populations due to geographical separation followed by migrations and mixing [29]. Also, as early as the 1960s, the book by MacArthur and Wilson [94] used models of geographical separation and migration to explain the spread and extinction of species. Thus, complete panmixia, although it can be achieved in the laboratory, appears only as a limiting case in nature, where spatial-separation effects seem to play an important role.

Today, it appears very clearly that topological structure largely determines the dynamical processes that can take place in complex systems. The study of both the structure and the dynamics of a given system is thus mandatory, if one wants to understand and possibly exploit all the possibilities. Why, thus, has evolutionary computation concentrated on mixing populations? One possible answer is that using a mixing population is both easier and good enough for many purposes and that the mathematical analysis of the dynamics of such populations is also usually easier. The next question is, thus: do spatially structured populations have a role in EC, i.e. are they a worthy object of study? I believe that the answer to this question should be positive. On the one hand, spatially structured populations are not new in EC: they have been much less studied than panmictic ones, but the empirical results available to date show beyond doubt that they can have beneficial effects. Secondly, there is actually only a very low cost associated with setting up and running a structured population instead of a panmictic one. Sometimes, as in the case of cellular populations, the cost of some genetic operations is even lower than that for a mixing population. So, experimenting with structured populations is generally easy and does not entail fundamental changes to the EC paradigm. On the other hand, spatially structured population models and their dynamical behavior are interesting objects in themselves; their interest goes beyond their mere utility as an empirical way of improving results in EC. This will be seen particularly in conjunction with some irregular population structures that have been found to be extremely significant in many fields.

Often, the main motivation for using structured evolutionary algorithms (EAs) is the possibility of physically distributing computational tasks over different machines and having these tasks executed in parallel. This is common practice, and is an advantage as far as computing times are concerned, especially for heavy-duty computations. However, the models and their implementations are two orthogonal factors. One may very well have a potentially parallelizable structured EA model and run it on standard sequential hardware. From the point of view of the model nothing changes, if the sequence of operations is the same, but of course the time to completion is likely to be less on truly parallel hardware. In this book I shall mainly describe models. Their actual implementation will be left unspecified in general, although some general comments on implementations will be given in an appendix to help practitioners.

Spatial or relational aspects are important in population dynamics, but the precise timing at which things happen also has an influence on the global population behavior. For this reason, I believe that space and time should be considered together, and this is precisely the point of view taken here. Thus synchronously and asynchronously evolving populations will be analyzed, especially with respect to cellular regular and irregular structures.

There is thus a double motivation for writing the present book. The first is the feeling that these considerations are indeed important, and the second is the fact that a broad, unified treatment of the field is lacking. Indeed, there exists an excellent book by Cantú-Paz on parallel genetic algorithms (GAs) [24]. But, as the title indicates, although the treatment is deep, the scope is rather specialized to master–slave and island GAs. Likewise, the several review articles that have appeared are useful, but they lack many details.

A word of caution is in order here. Spatially structured populations are relevant in many contexts, both in the natural sciences and in artificial models. It would have been an impossible task for me to try to give a decent account of all those different areas of investigation. Thus, it should be made clear that I only describe in this book structured populations as they are used in evolutionary algorithms. And even then, not all the families of EAs will be equally represented, for I shall concentrate on genetic algorithms and genetic programming. This last point is not a problem, for most considerations will extend naturally to other EAs. Other interesting families of EAs, such as multiobjective EAs and EAs with constraint handling are not dealt with directly. This is a pity, since most problems of practical interest are of this kind. However, I had to keep the length of the book to a reasonable size for a monograph, and many ideas will also be applicable, mutatis mutandis, to these more specialized EAs.

This book does not try to cover real-life applications either. There have been many applications of structured EAs in many fields but our focus is on models, theory, and the empirical properties of structured EAs. The interested reader will find an abundant sampling of successful applications of spatially structured EAs in the proceedings volumes of mainstream conferences such as the Genetic and Evolutionary Computation Conferences (GECCO), Parallel Problem Solving from Nature (PPSN), and the Congresses on Evolutionary Computation (CEC). Thus, the book is more suitable for advanced undergraduate and graduate students. However, it should also be useful for professionals and practitioners who would like to use structured populations in their applications of EAs.

Many other interesting objects of study, such as more general population-based artificial life systems, swarm intelligence models, natural-population, ecological, and social simulations, are not part of the book's subject matter either. In spite of this, several general conclusions that depend on the population structure will also teach us something valuable in apparently different contexts.

Another choice that had to be made concerns the level of acquaintance with EAs that is required for the reader to benefit from this book. It would have been nice to make the book self-contained by adding one or two introductory chapters on the main families of EAs. However, there are today several good introductory books on EC, some of which are listed in the references (e.g. [46], [99], [54]), or [145], if one looks for a more concise treatment. Thus, introducing this material with the necessary detail would duplicate existing work and would make the present book thicker and more awkward to write and read. Because of this, I have decided not to include such detailed introductory material, capitalizing on the assumption that the typical reader is likely already to possess a good knowledge of EC in general. However, if this is not the case, the present book will have to be supplemented with an introductory book on EAs.

Most books are the result of collective work, be it implicit or explicit, and this one is no exception. I have drawn from the work of many researchers who have paved the way for a better understanding of structured EAs in the last ten years. Closer to me, there are many colleagues who have contributed work, ideas, discussions, and suggestions. I should thus thank many people, but I would like to single out the following persons, for their help has been really outstanding. My former and present graduate students F. Fernández,

M. Giacobini, and L. Vanneschi, and my colleagues E. Alba, G. Spezzano, and A. Tettamanzi all contributed time and ingenuity to our common work on evolutionary computation using structured populations during the last few years. Just looking at the references will give an idea of my debt to all of them. M. Giacobini and L. Vanneschi also read several chapters and drew many of the figures. Other people helped in shaping my thinking about structured EAs, especially M. Sipper, who also made useful comments on Chap. 7.

Other people have provided support in several ways. In particular, I would like to thank Gusz Eiben, the Natural Computing Series Editor, for his encouragement and positive attitude toward this work. I am also deeply thankful to Springer's staff and Ronan Nugent, who provided invaluable help.

To all of them, and to the many who are not explicitly named, many thanks.

June, 2005
Marco Tomassini

Contents

1 Setting the Stage for Structured Populations

The aim of this initial chapter is to introduce some concepts related to structured populations in such a way that they are seen in the more general context of graphs or networks. With only a small cost in mathematical notation, this will allow us to discuss many apparently distinct features of populations under the umbrella of an existing and well-established mathematical notation.

Structured populations are just populations in which any given individual has its own neighborhood, which is smaller, sometimes much smaller, than the size of the population. In other words, instead of all the other individuals in the population being consirered as potential mates as in *panmictic* populations, only those that are in the same neighborhood can interact. Although this "isolation by distance" is often associated with geographical separation, this is not strictly required in evolutionary-algorithm (EA) models, where only the "relational aspect" matters. In fact, there are many examples of biological niches and isolated or semi-isolated populations in biology in which physical distance is the key factor keeping these *demes* nearly independent of each other. And many such biologically inspired models have been proposed for EAs. However, what counts is the neighborhood relationship, and this can be of any type, as long as it makes algorithmic sense. We shall see many examples of this in the following chapters. Thus, we are led to the conclusion that the important idea is the ensemble of relations among individuals, be they truly spatial or not. The mathematical objects that are required for describing this state of affairs are *graphs*. This means that we do not always need the concept of a metric space and an associated distance, such as the Euclidean distance. More often, distances between individuals will be given by the network itself, as measured along the path that links the two individuals in the graph. In these *relational graphs* the rules governing the construction of the graph do not depend upon any external metric between the vertices. For example, the electrical supply network of a given region would be a kind of spatial graph, since the distances between the graph nodes (generators, transformers and so on) are relevant, while the network of acquaintances in a society conveys only relationships, although other kinds of non-Euclidean distances can be associated with such networks. In the realm of population graphs, and with a slight abuse of language, I shall often call these population structures *topologies*, be they truly spatial or not.

Since graphs are a suitable mathematical description for structured populations, I shall give here a short introduction to the relevant concepts and definitions that will be used in the rest of the book. Graph theory is a well-developed branch of discrete mathematics and it would be impossible, and also useless, to try to give an account of it here, however brief. Instead, I shall limit myself to the introduction of the concepts that are really useful to us. In Chap. 6, we shall need a few more ideas about graphs. Since these concepts are not needed yet, I shall defer their presentation to the relevant place.

1.1 Useful Definitions for Graphs

For ease of reference, I collect here a few definitions and some nomenclature for graphs that will be used throughout this work. The treatment is necessarily brief: a more detailed account can be found, for example, in [22].

Let V be a nonempty set called the set of *vertices* or *nodes*, and let E be a symmetric binary relation on V, i.e. a set of unordered pairs of vertices. $G = (E, V)$ is called an *undirected graph*, and E is the set of *edges* or *links* of G. In a *directed graph* edges have a direction, i.e. they go from one vertex to another, and the pairs of vertices are ordered pairs. Figure 1.1 shows an undirected and a directed graph.

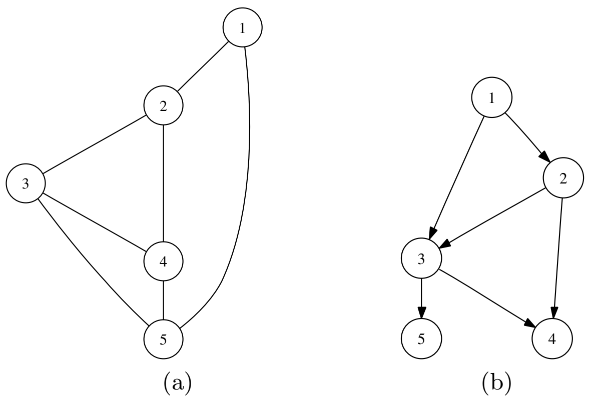


Fig. 1.1. (a) An undirected graph. (b) A directed graph

A *subgraph* of G is a subset of a graph's edges and associated vertices that constitutes a graph. That is, $G^{'} = (V^{'}, E^{'})$ is a subgraph of $G = (E, V)$ if $V^{'} \subseteq V$ and $E' \subseteq E$. For example, the set of vertices $1, 2, 4, 5$ in the graph G shown in Fig. 1.1 a induces a subgraph of G.

When vertices (u, v) of an undirected graph G form an edge, they are said to be *adjacent* or *neighbors*. The *neighborhood* of a vertex v is the set of vertices that are adjacent to v in G, not including v. For example, the neighbors of vertex 4 in Fig. 1.1 (a) are 2, 3, and 5. The *degree* k of a vertex is the number of edges impinging on it (or, equivalently, the number of neighbors). For directed graphs, one can correspondingly define the *outdegree* of a vertex v, which is the number of edges that leave v, and the *indegree*, which is the number of edges that enter the vertex v. The adjacency relation is not symmetric for directed graphs. For example, vertex 1 in the graph in Fig. 1.1 b has an outdegree of 2 and an indegree of 0.

A *path* from vertex u to vertex v in a graph G is a sequence of edges that are traversed when going from u to v with no edge traversed more than once. The *length* of a path is the number of edges in it. For example, the sequence $\langle 1, 2, 3, 5, 4\rangle$ is a path from vertex 1 to vertex 4. The *shortest path* between two vertices u and v is the path with the smallest length joining u to v. Thus, in Fig. 1.1 a, the sequences $\langle 1, 5, 4\rangle$ and $\langle 1, 2, 4\rangle$ are the two shortest paths from node 1 to node 4. Edges can have weights associated with them in some applications, and these weights are used in the calculation of the path lengths. If nothing is stated, each edge has a unit weight.

Cyclic paths are particular paths in a graph whose first and last vertices are the same. A *tour* is a particular cycle that contains every vertex. Again referring to Fig. 1.1 a, $\langle 1, 2, 4, 5, 1\rangle$ is a cycle, while $\langle 1, 2, 3, 4, 5, 1\rangle$ is a tour.

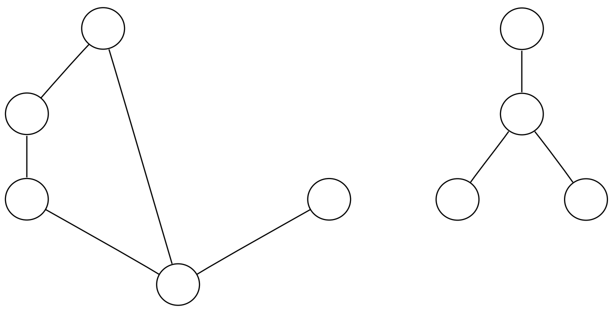


Fig. 1.2. An unconnected graph with two connected components

The maximum distance (path length) between any two connected vertices of a graph is called the *diameter* of the graph.

A graph is *connected* if there is a path between any two vertices. A graph that is not connected consists of a set of *connected components*, as illustrated in Fig. 1.2.

A *completely connected* undirected graph G with $|V| = N$ vertices has an edge between any two vertices. The total number of edges is thus $N(N-1)/2$. Figure 1.3 shows an example of such a graph with $N = 5$.

A *clique* in an undirected graph G is a completely connected subgraph of G.

A *sparse* graph has a number of edges $|E| \ll N(N-1)/2$. A *dense* graph has a number of edges $\propto N^2$.

A *star graph* is a network in which there is a particular node (the center) that is connected to all the other nodes, while the rest of the nodes are connected only to the center (Fig. 1.4).

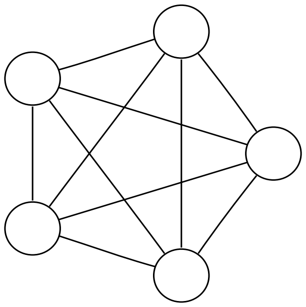


Fig. 1.3. A complete graph with five vertices

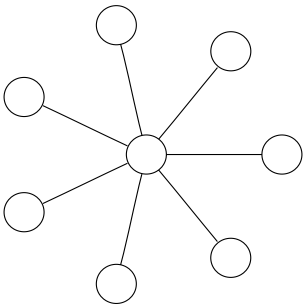


Fig. 1.4. A star graph

Finally, an *hypergraph* is like an undirected graph, but each edge connects an arbitrary subset of vertices and is called a *hyperedge*.

A graph G in which all the vertices have the same degree k is called k-*regular*. Complete graphs are obviously regular. Another important class of regular graphs for us is the family of *lattice graphs*. A *d-lattice* (or d-dimensional lattice) is an unweighted, undirected graph in which each vertex has the same degree k, with $k \geq 2$ and $k \geq 2d$. For example, with $d = 1$ and $k = 2$ and with $d = 1$ and $k = 4$, one obtains the ring structures shown in Fig. 1.5.

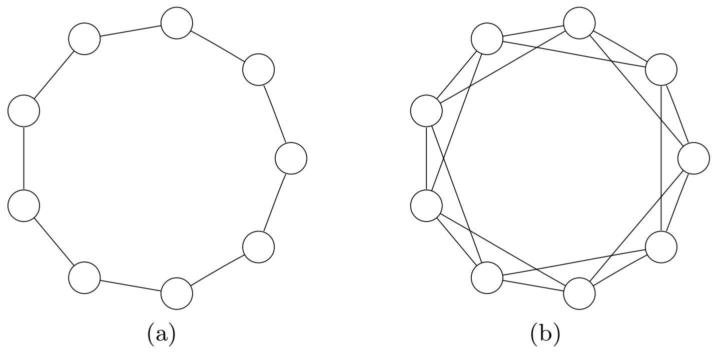


Fig. 1.5. (a) one-dimensional lattice with $k = 2$. (b) one-dimensional lattice with $k = 4$. Periodic boundary conditions are assumed

With $d = 2$ and $k = 4$ we have a torus, which is a topological entity with periodic boundaries (see Fig. 1.6).

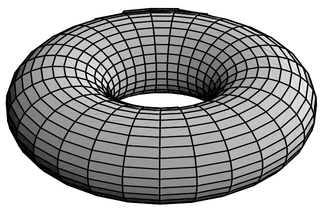


Fig. 1.6. A torus topology. This is obtained by wrapping the rows and columns of a two-dimensional grid around on themselves

Some of these graph types have been often used in evolutionary computation, and the dynamical properties of populations structured in these ways will be studied in detail in Chaps. 4 and 5.

1.2 Main Graph Structures of Populations

I shall now introduce the principal structures of populations that have been used in EAs, and these will be the base types for our analysis. Here I shall discuss only their graph-like properties. The dynamical evolutionary processes taking place in the populations will form the main theme of the book and will be dealt with in detail in the following chapters. Networks can be considered as the backbone on which dynamical processes take place. Therefore, networks are a prerequisite for describing the behavior of complex systems. Of course, the static and dynamic views are not divorced in reality, since there are mutual interactions between them. Think, for instance, of the communication processes taking place on a network that is itself continually changing, such as the Internet. However, our artificial structures are simpler, and it is useful to single out and consider their static structure first.

1.2.1 Island or Multipopulation Models

Here the idea is simply to divide a large panmictic population into several smaller ones. This model is usually called the *island* model or *multipopulation model*, and is schematically illustrated in Fig. 1.7. Each subpopulation runs a standard sequential EA, and individuals are allowed to migrate between populations with a given frequency. The migration directions are represented in the figure by arrows.

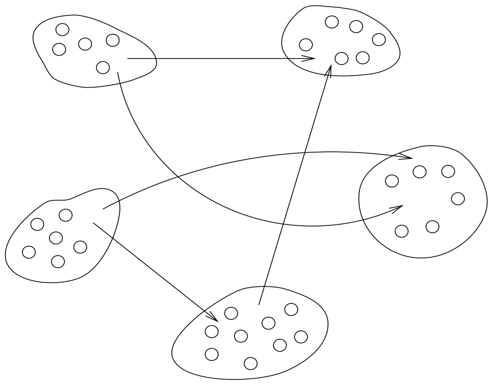


Fig. 1.7. A multipopulation structured model. Each "blob" represents a panmictic subpopulation. Subpopulations are loosely connected by periodically sending and receiving individuals according to the pattern shown by the arrows

In graph theory terms, each subpopulation is a vertex of the graph, and the edges are given by the migration links between islands. Note that the graph is usually a directed one. At a lower level, each island could be seen virtually, in turn, as containing a population structured as a complete graph. Several patterns of connection have traditionally been used. The most common ones are rings, two-dimensional and three-dimensional lattices, stars, and hypercubes. We shall deal with these models at length in Chaps. 2 and 3.

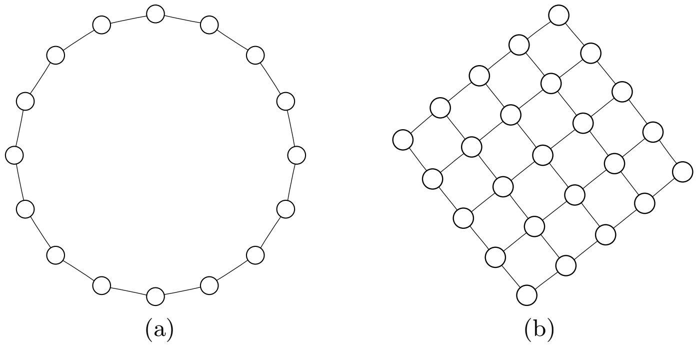


Fig. 1.8. (a) A one-dimensional-ring cellular population structure. (b) A two-dimensional-grid cellular population structure. In both cases each node is a single individual, and the edges wrap around in the grid case

1.2.2 Cellular Models

In *cellular* models, also called *diffusion* models, the individuals making up the population are usually disposed according to a regular lattice topology, i.e. a lattice graph (see Sect. 1.1). Two examples in one dimension and two dimensions, respectively, can be seen in Fig. 1.8, where the different shapes of the nodes represent potentially distinct individuals. In graph theory terms, each individual is a vertex of the graph, and edges link adjacent individuals, i.e. neighbors. In these cellular populations, each individual interacts only with a few other individuals in its neighborhood, and all genetic operations are local. Lattice-graph cellular populations will be examined in detail in Chap. 4. Of course, we are by no means limited to cellular models that are mapped onto regular lattices, although this has been the rule in the EA world. We shall see in Chap. 6 that many other graph topologies are possible and useful, including random and irregular structures.

Finally, it is worthwhile to make a comment on an interesting proposal of Sprave [141], who suggested using the hypergraph formalism to describe structured EA populations. Hypergraphs (see Sect. 1.1) are a generalization of simple graphs in which edges may span a subset of vertices. Hypergraphs can thus model any population structure, including the limiting panmictic case. Sprave's suggestion is attractive because it allows an elegant unified description of structured populations. However, for the sake of simplicity, and also because some new graph structures (see Chap. 6) are rather clumsy to represent with hypergraphs, I have chosen to stick with the more standard view of simple graphs.

1.2.3 Other Topologies

It is of course possible to design and implement population topologies more complex than the "basic" types described above. For example, one could have a multipopulation structure in which each subpopulation has a cellular topology. Alternatively, it would be possible to have a hierarchical island system, in which each island at an upper level contains a number of islands at a lower level. Exchanges would then be limited to taking place between islands at the same level.

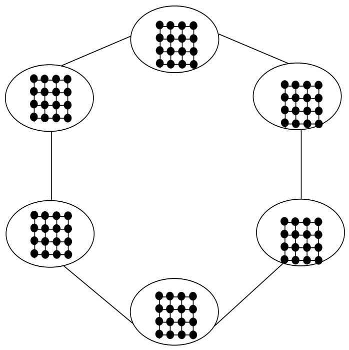


Fig. 1.9. A hierarchical EA model in which the populations in the loosely connected islands have a lattice structure.

Several other possibilities spring to mind as well. Figures 1.9 and 1.10 provide a schematic view of two *hierarchical*, or hybrid, population structures such as those described above.

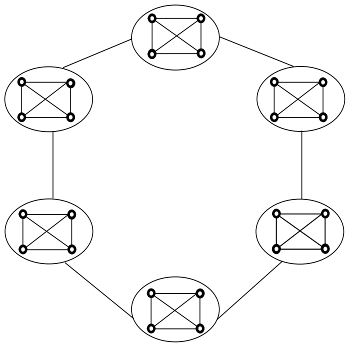


Fig. 1.10. A hierarchical EA model in which each island at the outer level contains a multipopulation EA. The communications between islands at the inner level are more frequent than the migrations at the outer-island level

Some of these structures, or analogous ones, have indeed been used in empirical work with good results. It would be tempting to try to extend the analysis that we shall do for the simpler cases to these more complex situations. However, I feel that our tools are not yet sharp enough to successfully deal with these more difficult cases (note than an analysis of a hierarchical master–slave EA model has been presented by Cantú-Paz in Chap. 8 of his book [24]). On the other hand, the simpler topologies already offer an extremely rich behavior and constitute a large field of investigation in themselves. As a consequence, I shall limit myself in the following to the description and analysis of "pure" island and cellularly structured population models.

2

Island Models

2.1 History and Background

This chapter is devoted to the island or multipopulation model, as it is closer to the original panmictic setting. As such, the corresponding EAs do not need a radical restructuring and are just simple variants of the standard ones.

As was observed in Chap. 1, the phenomenon of the formation of niches is common in biology. Niches impose mating restrictions, and thus the possibility of species differentiation. Islands and high, isolated valleys, for example, are the kinds of natural environments that can favor such evolutionary phenomena. Biologists have been aware of this for a long time, and Darwin himself commented on it. Although diversity tends to be low in each deme, overall population diversity is maintained through isolation. When other individuals have the possibility of interacting with previously isolated demes, interesting evolutionary processes may follow, such us colonization, extinctions, and punctuated equilibria, which are "jerky, or episodic, rather than a smoothly gradual, pace of change in evolution", in the words of Gould [49], one the fathers of the concept.

Thus, observations of real biological populations offer a number of ideas to evolutionary-computation researchers trying to improve the efficiency of their algorithms. In fact, these researchers were quick to take inspiration from the biological world as multipopulation EAs started to appear in the 1980s. We may cite, among others, the pioneering work of Grefenstette [73], Grosso [74], Tanese [144], and Cohoon et al. [32] on genetic algorithms. In the case of evolution strategies, Rudolph [123] implemented one of the first distributed models, and the work of Duncan [42] was an important milestone in evolutionary programming.

The purported advantages of island EAs are that they are supposed to explore a problem's search space more evenly and that they may fight population stagnation thanks to a better capability for maintaining overall diversity. We shall see that most empirical results to date, and some theoretical conclusions, tend to confirm these hopes.

The main idea of island EAs is to subdivide a single panmictic population of size N into s islands, or subpopulations, of smaller size n, such that $\sum_{i=1}^{s} n_i = N$. Of course, the size of a distributed population need not bear any particular relationship to that of a panmictic one; however, this view will be useful later when we compare the performance of island models with that of panmictic ones. Each subpopulation runs a standard EA as if it were isolated from the rest. However, from time to time, islands send and receive individuals to and from other islands. These groups of individuals, of size $m \ll n_i$, are often aptly called "migrants". The exchange can be made *synchronously* or *asynchronously* in time, in the sense that either all populations exchange individuals at fixed, predetermined times, or the exchanges may happen at independent times. The following pseudocode explains how a synchronous island EA works:

```
Initialize $s$ subpopulations of size $n$ each
$generation = 0$
while not termination condition do
    for each subpopulation do in parallel
        Evaluate and select individuals by fitness
        if $generation \bmod frequency = 0$ then
            Send $m$ best individuals to a neighboring population
            Receive $m$ individuals from a neighboring population
            Replace $m$ individuals in the population
        end if
        Produce new individuals using selection and variation operators
    end parallel for
    $generation = generation + 1$
end while
```

It appears from the above schemata that, in addition to the usual EA parameters, the model needs a few new ones: the number of subpopulations, the frequency of migration of individuals, the number of migrating individuals, and the communication topology. The existence of these additional degrees of freedom makes the algorithm more flexible, but also more difficult to control, since the parameters may interact in many ways. Unfortunately, there is no general theory to help us select suitable values for these parameters except, to some extent, in the case of genetic algorithms (GAs), as we shall see. To date, these parameters usually been set set empirically, building on the experience of previous work. Nevertheless, we shall see in Chap. 3 that there have been some systematic explorations of the parameter space that can be helpful in this respect.

Usually, the m migrating individuals are the top m in the original subpopulation, or chosen from among the best. Several individual-replacement policies have been described in the literature. One of the most common is for the m migrating individuals to displace the m worst individuals in the destination subpopulation. This is also the policy used in the simulations described here.

Several topologies for migration between islands can be used. The most common one is the "ring", in which populations are topologically disposed along a circle and exchanges take place between neighboring subpopulations, as illustrated in Fig. 2.1. Another possibility is "meshes of islands", possibly toroidally interconnected, in which migration occurs between nearest-neighbor nodes (see Fig. 2.2). One possible drawback of these *static topologies* is that some bias might be introduced by the repeating exchange pattern. *Dynamical topologies*, where destination nodes change with time, seem more useful for preserving diversity in the subpopulation. For example, good results have been obtained with the use of a "random" topology [50], where the destination population is randomly chosen at run time.

Different topologies have different *diameters* (see Sect. 1.1). The diameter of the graph contains an indication of the speed at which information will travel through the network, which is an important consideration for structured EAs, as we shall see. The shorter the diameter, the faster information will travel across the network. Rings have a large diameter $O(n)$, where n is the number of subpopulations, while square grids have a diameter $O(\sqrt{n})$. Other topologies, such as random graphs and small-world graphs (see Chap. 6) have a shorter diameter $O(\log n)$; the situation is the same for hypercubes. Completely connected graphs and star graphs obviously have a diameter of one. Section 3.6 presents empirical results on the influence of the topological arrangement of the subpopulations on the evolutionary process.

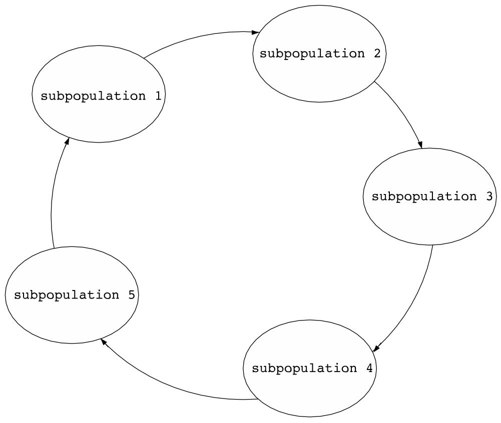


Fig. 2.1. Island model with ring topology. The subpopulations are arranged in a ring and individuals are exchanged between neighboring populations clockwise

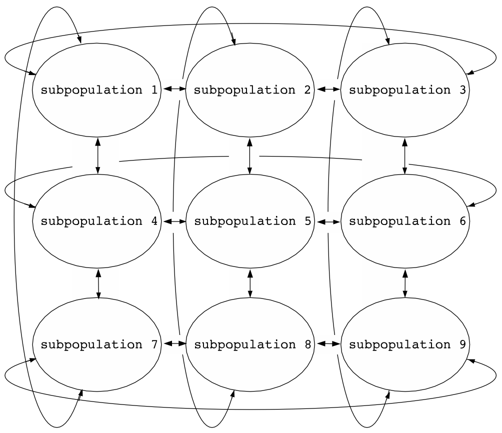


Fig. 2.2. Grid island topology. Populations communicate with their north, south, west, and east neighbors. The grid is folded into a torus

2.2 Homogeneous and Heterogeneous Islands

In the previous description it has been tacitly assumed that the EA parameters, the subpopulation size, and the representation of individuals are the same in all the islands. If this is the case, the EA is called *homogeneous*. This is the simpler and customary choice, but it need not always be the case. Indeed, it is possible, and sometimes useful, to have different parameters and/or different representations in different islands. It is also possible, as in some panmictic EAs, to allow for islands of variable size. We shall call these island models collectively *hetereogeneous* and *nonstandard*. In this chapter and the two that follow, we shall deal with homogeneous island models exclusively. In fact, these models are wellknown and easy to implement, and they usually give good results, as we shall see in the following chapter. Moreover, there is a fair amount of theoretical results for them, which is not the case for the heterogeneous models. However, we cannot ignore heterogeneous models only on the grounds that they are difficult to understand, since good results have been obtained in practice. Thus, in Chaps. 7 and 8, I shall present some of these models.

2.3 Theoretical Results

The early work on island EAs was largely empirical in character. There was a general feeling that sparsely communicating subpopulations were advantageous in terms of problem-solving capabilities, and they also saved computing time, when implemented on parallel or distributed hardware. However, the reasons why the models were efficient were largely unknown. It was also unclear how one should choose the several new parameters that were needed. Another important question is the following: what are the problem classes that benefit most from a distributed population setting and why?

Thus, a basic understanding of the workings of island EAs is necessary but, even today, there is no clear-cut answer to the questions above. Some progress has been made, however, especially by Cantú-Paz for binary-coded GAs as described in his book [24] and the articles mentioned therein. Before describing Cantú-Paz's main results, I shall briefly comment on early investigations aimed at understanding the main dynamical aspects of multipopulation EAs.

One of the first attempts to theoretically analyze an island model GA was published by Pettey and Leuze [116]. In this work, a schema theorem was derived for an island system in which individuals chosen uniformly at random are sent to all other populations in each generation. Pettey and Leuze proved that this model maintains the same efficiency as does a panmictic-population GA, and that the expected number of trials can be bounded from above by an exponential function, i.e. the same global behavior that is observed for panmictic GAs. While the analysis did not take into account the effect of varying parameters such as the island size, the migration rate, and the migration frequency, it was a useful first step.

More recently, Whitley and coworkers asked the following important question: under what conditions is a given problem suitable for an island model? In other words, are there features of a problem that allow a multipopulation EA to solve the problem more efficiently (in the algorithmic sense) than does a panmictic EA? The issue was first raised in [143], and it was given a partial answer in [159]. Whitley et al. used Vose's infinite-population model to hint at the fact that multiple islands are more likely to maintain diversity and produce new solutions, thus showing superior search performance compared with single-population models. Then Whitley et al. studied the particular class of *linearly separable problems*, i.e. those in which the objective function F is a sum of s independent nonlinear subproblems G_i: $F = \sum_{i=1}^{s} G_i$. Using a probabilistic argument, they built a simple model that was then tested on two linearly separable functions: the fully deceptive order-4 function and the Rastrigin function. Using their steady-state distributed version of the GENITOR software, they performed 30 runs for each problem and each combination of parameters. The parameters were the number of subpopulations, the migration rate, the migration frequency, and the presence or absence of mutation. Crossover was always used. They found that larger populations were useful for the Rastrigin function, but did not help much in the deceptive problem. The conclusion was that islands may be useful when increasing the total population size does not help to solve the problem. They found it difficult to generalize the conclusions, owing to the interplay of the many parameters and the complexity of the process. Among other things, the optimal solution has a nonnegligible chance of being contained in the initial populations, for large population sizes, which of course may spoil the results. Nevertheless, although many questions remained unanswered, this study has shed some light on the behavior of island models.

As I said above, the recent work of Cantú-Paz on GA island models is the most thorough and quantitative to date in the field. Here I shall only highlight the main results. For the details, the reader is referred to Cantú-Paz's excellent presentation in [24].

Cantú-Paz and Goldberg [25] extended Goldberg's previous work [66] on the sizing of single populations for multiple islands. Of course, the size of the populations is the single most important factor that will influence the behavior of the collective system: islands that are too small will fail to provide enough diversity and will converge prematurely. On the other hand, if the islands are too large with respect to a single population that solves the problem, extra useless work will be performed. Cantú-Paz and Goldberg focused on modeling the speedup offered by the multipopulation system for reaching a solution of the same average quality as the one found by use of the serial process. They treated two important bounding cases of multipopulation GAs: a set of completely connected islands, and a set of isolated islands. In the complete-graph case, the migration rate and frequency were set to the maximum value, i.e. every island exchanged individuals with every other island in each generation. The main conclusions were that the speedup with respect to the single panmictic population was better for the communicating system than for the isolated model. Cantú-Paz and Goldberg also found that, for these limiting cases, there was an optimal number of islands and a corresponding island size that maximized the speedup.

Chapters 5 and 6 of Cantú-Paz's book [24] extend the analysis to arbitrary topologies, an arbitrary number of islands, arbitrary sizes, and arbitrary migration rates. Markov chain methods are used for modeling the evolutionary process, and the accuracy of the models is tested experimentally on fully deceptive functions by varying the parameters of interest. The observation is that the solution quality improves with higher migration rates and denser topologies. However, the communication cost tends to increase in this case, and there is a trade-off between solution quality and algorithm speed. It is also shown that different topologies with the same number of neighbors per island reach almost equivalent solutions. We shall see in the next chapter that experimental results tend to qualitatively confirm these model predictions. With a fixed topology, the optimal number of islands is found to be

$$O\left(\sqrt{\frac{nT_f}{T_c}}\right),$$

where n is the population size, T_f is the time required to evaluate an individual, and T_c is the mean time required to communicate with another deme.

2.4 Asynchronous Islands

Timing considerations will come up often in this book, for the temporal sequence of operations can make a notable difference to the resulting dynamical processes. We shall see that these ideas have more significant consequences in the case of cellular systems in Chaps. 4 and 5. But it is worth introducing some concepts of synchronous and asynchronous evolution for multipopulation models too. Essentially, synchronous systems work in lockstep as if they were reacting to the ticks of a global clock. Of course, there can be different degrees of lockstep. In computer hardware, for example, in each clock cycle one or more elementary operations are executed. In island models, which are rather coarse-grain, there is more freedom: the islands evolve independently for a certain time, then they synchronize and send and receive a certain number of individuals. At the end of this "handshaking" phase, the islands are computed again in an independent manner. This description corresponds essentially to the pseudocode of Sect. 2.1. Of course, synchronous send and receive implies that everything must wait for the slowest process to finish the current generation before the exchange takes place. Although we are not talking about computer implementations here, it is worth noticing that this idle wait might cause a synchronization overhead whenever the model is implemented on physically replicated hardware. In practice, most EAs take on average the same time to process the same number of individuals, and thus the differences are small, except in the case of variable-length-representation EAs, such as in genetic programming.

Asynchronous systems, on the other hand, do not have a global clock and need not synchronize at specified points of the computation. In Chap. 4 we shall study an extreme case of this: cellular EAs in which each individual is updated at arbitrary times. Here we consider a coarser-grain model of asynchronism based on communicating islands. In this case, the algorithm is essentially unchanged except for the fact that sending and receiving individuals may be done at independent times: an island sends a boat of migrants at some point in time, and the destination island gets the individuals when it is ready to do so.

Asynchronous operation may simply arise naturally from the fact that, perhaps, different islands take different times to evaluate their individuals and the system does not wait for the slowest island to reach the synchronization point. Asynchronous communication between islands may also be the result of a deliberate algorithmic choice. For example, Munetomo et al. [109] have suggested that the exchange of migrants may be triggered by each subpopulation by measuring a parameter related to its fitness distribution and phenotypic diversity.

In these asynchronous algorithms, after a certain time, the different islands will be working in different generations. To some extent, the situation is similar to what happens in *steady-state* EAs, where the whole population is not changed at once but rather only a part of it, and successive generations have a degree of overlap. Lin et al. [93] have presented a study of the various possibilities and of their possible advantages and drawbacks.

Asynchronous island models are more difficult to analyze theoretically but have often been found useful in applications. Examples of asynchronous island EAs and their empirical characterization will be discussed at the end of the next chapter.

3

Island Models: Empirical Properties

3.1 An Experimental Investigation

In the preceding chapter we have seen how the island model EA is structured and how it works. Some theoretical results have also been presented. In the present chapter I shall try to provide empirical evidence for the practical usefulness of the model.

The island model has been very popular among EA researchers because it is easy to implement and usually gives satisfactory results. There have been many empirical investigations and applications, some of which are listed in the references (see, for instance, [7, 8, 11, 93] and the recent review [5]).

In order to illustrate the main features of island EAs, I shall present here a rather systematic case study using genetic programming (GP) that originally appeared in [51] and to which the reader is referred for more details. Fernández and Vanneschi's PhD theses also contain a wealth of information on the topic [39, 150]. Readers unfamiliar with GP might consider consulting [17] or [89]. The case study will be integrated with an investigation of the behavior of population diversity, and with a comparison between synchronous and asynchronous island models.

3.2 Description of the Test Problems

The choice of test problems in an empirical investigation is always a difficult task. Ideally, one would like to consider many problems of variable difficulty and belonging to different classes, including real-life problems. However, time and computational resources are limited. This is why, usually, only a few problems are included in the benchmark. Genetic programming makes the situation even worse, since it is in general more time-consuming than other EAs.

Here I address a set of problems chosen from those that have been classically used for testing GP: the even-n-parity problem, the symbolic regression problem, and the problem of the artificial ant on the Santa Fe trail, since there is a large amount of accumulated knowledge on these in the GP community ([89, 92]). Results on real-life problems can be found in [51].

The following is a brief description of the test problems. For more detailed explanations, see [89].

Even-Parity-4 Problem

The boolean even-parity-k function of k boolean arguments returns *true* if an even number of its arguments evaluates to true, otherwise it returns *false.* If $k = 4$, then 16 fitness cases must be checked to evaluate the fitness of an individual. The fitness is computed as 16 minus the number of hits for the 16 cases. Thus a perfect individual has fitness 0, while the worst individual has fitness 16. The set of functions is the following: $F = \{NAND, NOR\}$. The terminal set in this problem is composed of four different boolean variables $T = \{a, b, c, d\}$.

Artificial Ant on the Santa Fe Trail

In this problem, an artificial ant is placed on a 32×32 toroidal grid. Some of the cells in the grid contain food pellets. The goal is to find a navigation strategy for the ant that maximizes its food intake. The same set of functions and terminals as in [89] was used in the study described here. The fitness function is the total number of food pellets lying on the trail (89) minus the amount of food eaten by the ant during its path. This turns the problem into a minimization one, like the previous one.

Symbolic Regression Problem

The aim here is to find a program which matches a given equation. The classic polynomial equation $f(x) = x^4 + x^3 + x^2 + x$ is used, and the input set is composed of the values 0 to 999 (1000 fitness cases). For this problem, the set of functions used for GP individuals is the following: F={*, //, +, -}, where // is like / but returns 0 instead of *error* when the divisor is equal to 0, thus allowing syntactic closure. The fitness is defined as the sum of the squared errors at each test point. Again, a lower fitness means a better solution.

3.3 Multipopulation GP Parameters

In all the experiments, the same set of GP parameters was used: generational GP, crossover rate 95%, mutation rate 0.1%, tournament selection of size 10, ramped half-and-half initialization, maximum depth of individuals for the creation phase 6, maximum depth of individuals for crossover 17, and elitism (i.e. survival of the best individual into the newly generated population for panmictic populations; the same was done for each subpopulation in the distributed case). Furthermore, to avoid complicating the issue, I refrained from using advanced techniques such as ADFs, tailored function sets, and so on. Only plain GP was used in the experiments.

As already noted, a number of new parameters must be considered in the island model of parallel, distributed GP:

- the number of populations,
- the number of individuals per population,
- the communication topology,
- the number and type of migrating individuals, and
- the frequency of migration.

The main difficulty faced by the experimenter is that the simultaneous setting and tuning of all the parameters is a monumental, and probably hopeless, task. In practice, tuning is done one parameter at a time, which gives less than optimal results since the parameters are interdependent and interact in complex ways. However, this is a feasible task, and it allows one at least to investigate suitable parameter ranges for a series of problems, and the qualitative ways in which some of the parameters interact with each other.

On the basis of some theoretical results for multipopulation GAs [24], it is assumed that the number of populations and the number of individuals per population are more important and are studied first, keeping the other parameters fixed at reasonable values on teh basis of previous empirical knowledge. Once a satisfactory set of values has been obtained for the fundamental parameters, finding good values for the others becomes possible by means of extensive simulations.

3.4 Performance Measures and Statistics

For the kind of experiment described here, whose aim is to understand what are the conditions that make island EAs more efficient than panmictic EAs, we are not interested so much in obtaining perfect solutions for each problem by using special parameter settings, special techniques for maximizing performance, or special knowledge about the problem. Instead, we are concerned with the dynamics of evolution and how the convergence process takes place in the distributed case, as compared to standard GP, over many runs, in order to find statistical regularities and robust and reproducible behavior. To compare efficiency, we need a performance measure. Usually, the number of fitness evaluations is an acceptable metric for EAs, since all the other parts of the algorithm either have negligible complexity or have the same cost for different runs of the same problem. However, this metric is inadequate for GP, in which, owing to the variable-length representation, individuals may have widely different complexities during evolution.

The data are thus analyzed by means of the *effort of computation*, which is defined as the total number of nodes that GP has evaluated in a population for a given number of generations. Note that, strictly speaking, one should take into account the fact that evaluation times may be different for different operators, but this simplification is still useful as a first approximation. Clearly, this measure is problem-specific, but it is useful for comparing different solutions of the same problem.

One last remark about performance evaluation is that it is not my intention here to compare actual execution times. Island models can be run on several machines simultaneously, which can provide quite good speedups, since the communication phases are short with respect to computation. However, as already noted, I wish to concentrate on the merits of the models themselves as new kinds of evolutionary algorithms, and not on their time efficiency.

Reporting on performance comparisons in the field of EAs is a notoriously difficult issue. For problems for which the solution is not known, such as hard real-life optimization problems, a useful figure of merit is the mean best fitness (MBF), that is, the average over all runs of the best fitness values at termination [47]. The very concept of termination is a fuzzy one. Indeed, in the above situation, one does not know in advance whether the global optimum has been reached. Consequently, one common approach is to take the measure after a specified amount of computational effort. For problems with known solutions, such as those that are studied here, the above measure is not entirely adequate, because a sizable part of the runs are unsuccessful for the prescribed effort (using a larger effort would help in some cases but would become prohibitively expensive). This prevents one from knowing whether increasing the length of the runs would have been useful and in which cases. Thus, the measured MBF says little about the problem-solving capabilities of methods. Instead, when the solution is known, the success rate (SR), defined as the ratio of successful to the total number of runs for a specified computational effort, is a good indicator of the algorithmic effectivity [47].

Thus, the main performance indicator used here is the SR. However, to give a more complete picture of the whole evolutionary process, we also present mean fitness curves against computational effort, and fitness histograms giving the relative frequency of the solutions found at a given effort value. The mean fitness curves are useful for getting a visual feel for the workings of different algorithms with respect to each other over time. The histograms allow one to know not only the number of perfect solutions but also how many solutions were close to optimal, which gives an idea of the "dispersion" of the solutions at the end of the runs.

From the statistical point of view, a series of runs of a problem may be considered as a series of independent Bernoulli trials of the same experiment having only two possible outcomes: success or failure. In this case, the number of successes (or failures) is binomially distributed and the maximum-likelihood estimator $\hat{p}$ of the mean of a series of Bernoulli trials, and hence for the probability of success, is simply the number of successes divided by the sample size (the number of runs n). With this information, one can calculate the sample standard deviation $\sigma = \sqrt{n\hat{p}(1-\hat{p})}$, which is also given in the tables.

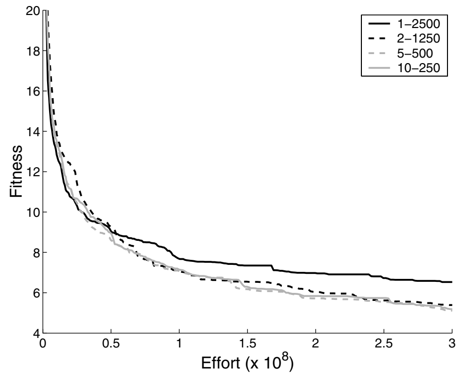
3.5 Number of Subpopulations and Their Size

In this section two of the parameters affecting the performance multipopulation GP are studied, namely the number of populations and their size. The interesting limiting case of isolated subpopulations is presented first, and then communication, and thus migration, is introduced. Numerical experiments, described in [51], allow one to find suitable population sizes, or rather size regions, for each test problem such that using larger populations is useless or even harmful. Using smaller populations, on the other hand, may cause the search to fail to converge most of the time. Therefore, the number of individuals found in those numerical experiments will be used in what follows.

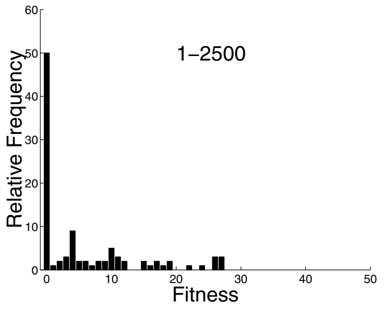
All the curves that appear in this section are averages over 100 independent runs of the same experiments and are labeled by two integers p and n, where p is the total number of populations and n is the number of individuals contained in each population. Thus, for example, "1–2500" means a single population of 2500 individuals (standard GP), while "5–500" means five populations with 500 individuals per population. The runs have a maximum length of 500 generations, and the maximum effort in each case was fixed so as to be achievable within this limit. Here I only show some of the results for simplicity, and to avoid bothering the reader with too much data and graphics. This should be enough to demonstrate some general lessons. The full data set is discussed in [51].

3.5.1 Isolated Populations vs. Standard GP

The distinctive aspect of island GP is the exchange of individuals among subpopulations. A bounding case arises when the populations evolve in isolation. This bounding case has been analyzed by Cantú-Paz for GAs [24]. A single population of N individuals can be subdivided into n subpopulations which run in parallel on $m \leq n$ machines or sequentially on a single processor because the runs are independent. The best fitness is obtained by selecting the best individual from the subpopulations at the end of the runs. Obviously, this population structure does not require any change to the standard GP algorithm. On the other hand, the number of individuals in each subpopulation is lower than that used when only one population is employed. Consequently, the search space region explored by a subpopulation is smaller than that explored by the whole population. Therefore, the time gains offered by the distributed system might be offset by an insufficient quality of the evolutionary search. But if a given number of individuals, say l, happens to be enough for solving a given problem, larger populations, which need more computational effort, represent a waste. A question thus arises: is it better to use several smaller subpopulations or a larger population with the same total number of individuals? The answer depends on the nature of the search space and on the size of the populations used for the search.



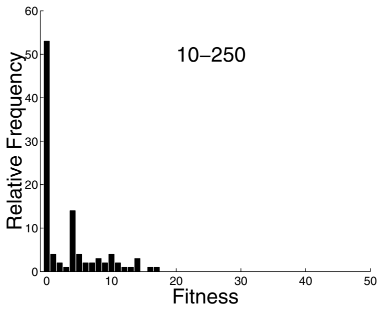
(a)



(c)

	E = 1x10^8	E = 2x10^8	E = 3x10^8
1- 2500	43	47	50 *(σ = 5.000)*
2- 1250	46	49	52 *(σ = 4.996)*
5- 500	51	53	55 *(σ = 4.975)*
10- 250	50	50	53 *(σ = 4.990)*

(b)



(d)

Fig. 3.1. The artificial ant problem. Standard GP vs. IMGP. (a) Average fitness against effort for 1, 2, 5, and 10 populations and a total number of 2500 individuals over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found by standard GP for an effort $E = 3 \times 10^8$. (d) Histogram of the relative frequency of the solutions in the case of 10 populations of 250 individuals each, for the same value of the effort

Artificial Ant Problem

Figure 3.1 shows that, for a total number of individuals equal to 2500, isolated-multipopulation GP (IMGP) performs better than standard GP, although the standard deviations on the success rates (Fig. 3.1 b) indicate that the improvement may be only marginal. IMGP with two, five, or ten populations gives the best results, as depicted in the effort curves (Fig. 3.1 a), while using one single population slightly decreases the performance. The same effect is apparent in the histograms of the relative frequency of the solutions, where it can also be seen that even when perfect solutions are not found, better solutions tend to be discovered in the multipopulation case. With smaller populations (500 individuals, results not shown here to save space), the general trend is similar. However, the best average fitness reached with 500 individuals is worse, as is the number of hits in all cases. This is because the ant problem is a difficult and deceptive problem for GP (see [92] for an in-depth analysis), and 500 individuals represent too small a sample for this problem.

Even-Parity-4 Problem

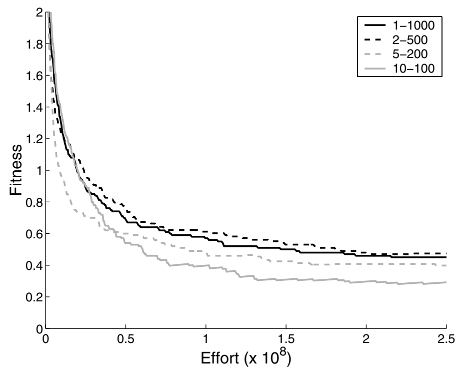
Figure 3.2 illustrates results obtained on the Even-Parity-4 problem with a population size of 1000 individuals. Even-Parity-4 instead of the more usual Even-Parity-5 was chosen because the latter is more difficult, requiring larger populations and more computing time, and its low success rate cannot provide useful data for statistical analysis. For a total population size of 1000, the general trend is that isolated populations are more efficient than a single large population. The standard deviations (Fig. 3.2 b) in the case of the 1–1000 and 10–100 experiments show that the difference in success rate is significant. This behavior is qualitatively confirmed by the average fitness graphs. It appears that ten populations of 100 individuals each represent a good choice, while two populations of 500 individuals perform slightly worse. The histograms of the relative frequency of the solutions for standard GP and for ten populations confirm that the number of perfect solutions found in the multipopulation case is always higher, and even when perfect solutions are not found, the distributed case tends to find better individuals.

However, results not shown here indicate that there is no clear advantage for the multipopulation case with a total of 500 individuals, because the subpopulations become too small, although the distributed case offers the benefit of a shorter computation time if the runs take place simultaneously.

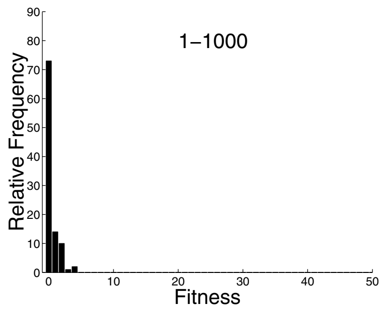
The number of populations is a crucial issue: we must balance it against the number of individuals per subpopulation in order to carry out an effective search and, on the other hand, explore different areas of the search space.

Symbolic Regression Problem

Figure 3.3 depicts results obtained by means of standard GP and with isolated populations for the symbolic regression problem, for 100 GP runs with a total population size of 250. The figure shows that isolated populations have a slight edge, although the effects of distribution have only marginal statistical significance, as judged by the standard deviations on the success rates (see Fig. 3.3 b). Moreover, note that the symbolic regression problem used here (small degree, no adjustable constant coefficients) is easier than either the ant or the even-parity problem (i.e. a higher number of individuals with a



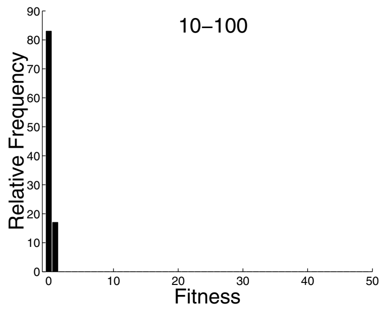
(a)



(c)

	E = 1.5x10^8	E = 2x10^8	E = 2.5x10^8
1-1000	71	72	73 *(σ = 4.439)*
2-500	64	68	70 *(σ = 4.582)*
5-200	76	77	77 *(σ = 4.208)*
10-100	76	81	83 *(σ = 3.756)*

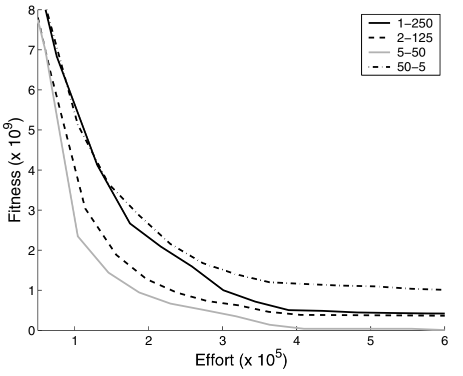
(b)



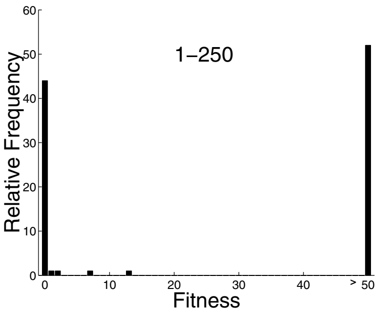
(d)

Fig. 3.2. The even-parity-4 problem. Total population size 1000. Standard GP vs. IMGP. (a) Average fitness against effort for 1, 2, 5, and 10 populations and a total number of 1000 individuals over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found by standard GP for an effort $E = 2.5 \times 10^8$. (d) Histogram of the relative frequency of the solutions in the case of 10 populations of 100 individuals each, for the same value of the effort

perfect fitness is found with less computational effort). The trend continues with smaller populations, but there is a limit on the number of individuals that can do a good job. For instance, note how the extreme case of 50 populations containing only five individuals each shows a degradation in performance in Fig. 3.3 a. Experiments with a total number of 500 individuals (not reported here) show the same behavior, except that the multipopulation case is more favorable. This is probably because 500 is a large population size for this problem, and dividing the population into several smaller ones still produces subpopulations that are large enough to allow us to find good solutions; in other words, a single population of 500 individuals expends more effort than necessary for obtaining good-quality solutions.



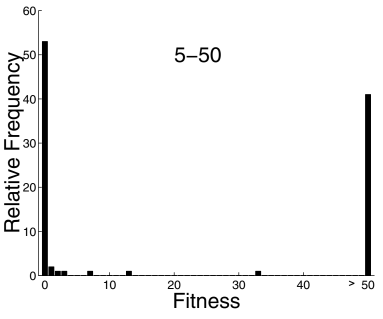
(a)



(c)

	E = 2x10^5	E = 3x10^5	E = 4x10^5
1- 250	38	44	44 *(σ = 4.964)*
2- 125	41	46	47 *(σ = 4.990)*
5- 50	50	52	53 *(σ = 4.990)*
50- 5	36	41	42 *(σ = 4.935)*

(b)



(d)

Fig. 3.3. The symbolic regression problem. Total population size 250. Standard GP vs. IMGP. (a) Average fitness against effort for 1, 2, and 5 populations and a total number of 250 individuals over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found in by standard GP for an effort $E = 4 \times 10^5$. (d) Histogram of the relative frequency of the solutions in the case of 5 populations of 50 individuals each, for the same value of the effort. In the histograms, the height of the bar marked "> 50" is proportional to the number of solutions that are at least 50 units of fitness worse than the perfect solution

3.5.2 Communicating Islands vs. Standard GP

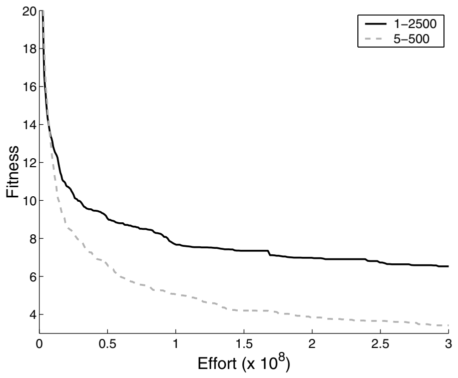
The isolated-population situation is an interesting limiting case, but one of the purported advantages of island GP is the possibility of exchanging information between the subpopulations. Thus, the next logical step is to introduce communication, i.e. migration of individuals between demes.

The important point here is, again, the number of subpopulations and their size. In order to limit the number of free parameters, the other parameters were chosen as follows:

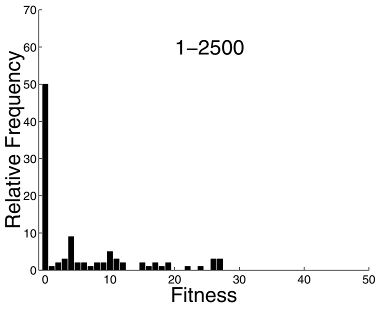
- communication topology: random;
- frequency of exchange: every ten generations;
- number of individuals exchanged: 10% of the population size.

These values are not arbitrary: they have been used before in empirical work with good results and should form an acceptable first approximation. Their variation is studied later in Sects. 3.6 and 3.7, where it will be seen that this choice is a reasonable one.

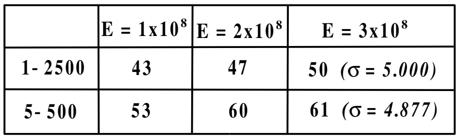
Artificial Ant Problem



(a)

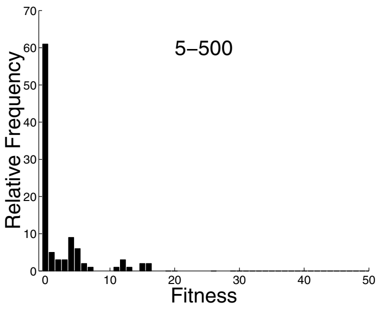


(c)



	E = 1x10⁸	E = 2x10⁸	E = 3x10⁸
1-2500	43	47	50 *(σ = 5.000)*
5-500	53	60	61 *(σ = 4.877)*

(b)



(d)

Fig. 3.4. The artificial ant problem. Standard GP vs. island GP with random communication topology (see text for the other communication parameters). (a) Average fitness against effort for 1 and 5 populations and a total number of 2500 individuals over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found by standard GP for an effort $E = 3 \times 10^8$. (d) Histogram of the relative frequency of the solutions in the case of 5 populations of 500 individuals each, for the same value of the effort

Figure 3.4 clearly shows that five communicating populations of 500 individuals each are more effective than a single population of 2500 individuals. In fact, both the curves of average fitness vs. computational effort and the table of the number of hits show better performance for the distributed case, as confirmed by the standard deviations; the relative-frequency histograms show that the distributed case finds solutions of better quality.

The case of five communicating populations can also be compared with the case of five isolated populations for the same problem (see Fig. 3.1). We observe that the average fitness curve goes below a value of 4 when communication is allowed, whereas it only reaches a value near 6 in the isolated case. Moreover, the number of successes is higher when populations communicate, and the histogram of the relative frequencies of solutions shows that more solutions of better quality are found when communication is allowed. The differences are significant in the case of island GP vs. standard GP, while they are not significant for the isolated-population case. This observation refers to a fixed value of the computational effort, which is the same in both cases. Clearly, if it were given more generations to run, the isolated case could eventually reach the same performance level but the effort would be larger.

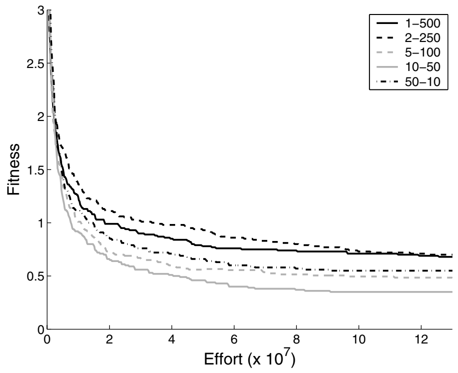
Even-Parity-4 Problem

In Fig. 3.5, we can see again that island GP with a random topology performs significantly better than standard GP for all the population sizes reported in the figure. In particular, distributing the individuals into ten populations gives the best results, as is apparent from both the average-fitness curves and the number-of-hits table. The relative-frequency histograms confirm the trend, with a higher number of perfect solutions found and a clustering of good solutions that is shifted toward better fitness values in the distributed case. The trend favors smaller multiple populations up to a point. However, if we use too small a population size, the number of islands and the migration cannot compensate for the loss of search power, as can be seen in Fig. 3.5 for the extreme case of 50 populations of 10 individuals each, where performance degrades.

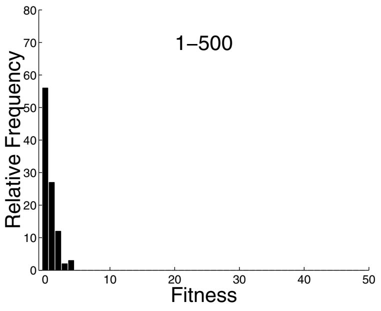
Symbolic Regression Problem

Figure 3.6 depicts the results obtained for the symbolic regression problem with 250 individuals. Again, we see that distributing the individuals into five or ten populations is beneficial for this problem, as shown by the number of successes and the standard deviations. With respect to the analogous isolated-population case, one can say that the results are almost the same, which can be explained by the simple nature of the problem. At any rate, the multipopulation case is more favorable than standard GP for both communicating and isolated populations in this case.

The tables in Fig. 3.7 summarize the differences between standard GP, isolated-multipopulation GP (IMGP), and communicating-island GP for the



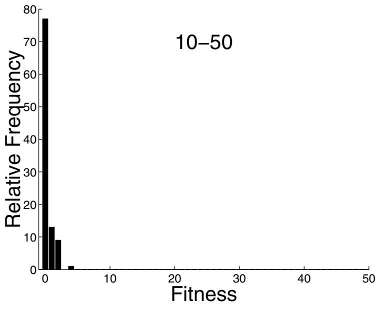
(a)



(c)

	E = 8x10^7	E = 10x10^7	E = 12x10^7
1- 500	53	54	56 *(σ =4.964)*
2- 250	55	60	60 *(σ =4.899)*
5- 100	64	65	65 *(σ =4.770)*
10- 50	76	77	77 *(σ =4.208)*
50- 10	62	64	65 *(σ =4.770)*

(b)



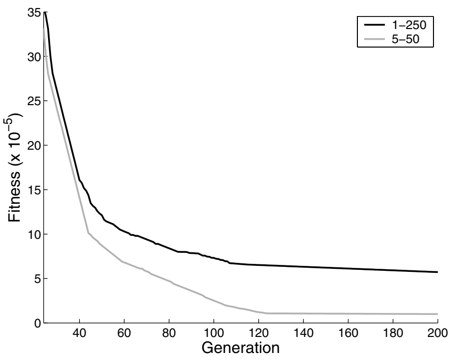
(d)

Fig. 3.5. The even-parity-4 problem. Standard GP vs. island GP with random communication topology (see text for the other communication parameters). (a) Average fitness against effort for a total number of 500 individuals over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found by standard GP for an effort $E = 12 \times 10^7$. (d) Histogram of the relative frequency of the solutions in the case of 10 populations of 50 individuals each, for the same value of the effort

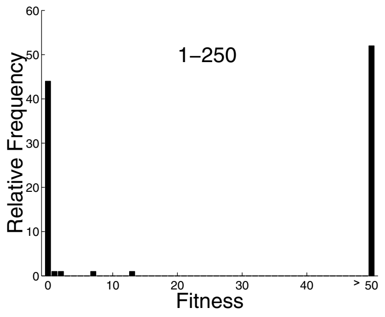
test problems, for some typical population sizes and numbers of subpopulations. One can see that the differences between island GP and standard GP are always significant, while this is not always the case for IMGP compared with standard GP. Migration thus offers a significant advantage, at least for the test problems used here.

3.6 Comparing Communication Topologies

Now we investigate the influence of island communication topology on the evolutionary process, one of the main themes of this book. The topologies used here are the ring (or "circle"), the two-dimensional grid, and the random



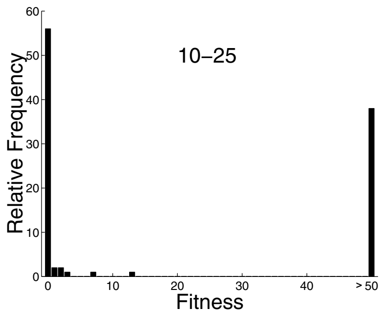
(a)



(c)

	E = 2x10^5	E = 3x10^5	E = 4x10^5
1- 250	38	44	44 *(σ =4.964)*
5- 50	51	54	55 *(σ =4.974)*
10- 25	53	55	56 *(σ =4.964)*

(b)



(d)

Fig. 3.6. The symbolic regression problem. Standard GP vs. island GP with random communication topology (see text for the other communication parameters). (a) average fitness against effort for a total number of 250 individuals over 100 executions. (b) number of successes in 100 runs for three values of the total effort. (c) histogram of the relative frequency of solutions found by standard GP for an effort $E = 4 \times 10^5$. (d) histogram of the relative frequency of the solutions in the case of 10 populations of 25 individuals each, for the same value of the effort. In the histograms, the hight of the bar marked "> 50" is proportional to the number of solutions that are at least 50 units of fitness worse than the perfect solution

topology (see Sect. 3.1). The rings considered are *one-way*, i.e. messages always travel from a population to a neighboring one in the same direction. In the two-dimensional grid, messages are sent from an island to its four south, north, west, and east nearest neighbors. On the other hand, in the random topology, the destination population is chosen randomly at run time. All the results described in this section have been obtained by averaging 100 independent runs of the same experiment. A total number of individuals equal to 10% of the total population size was exchanged between populations every 10 generations. Figure 3.8 shows results for the artificial ant problem in the case of nine populations of 50 individuals each.

	1-2500	IMGP 5-500	PADGP 5-500
Artificial Ant	50 *(σ=5.000)*	55 *(σ=4.975)*	61 *(σ=4.877)*

	1-1000	IMGP 10-100	PADGP 10-100
Even Parity 4	73 *(σ=4.439)*	83 *(σ=3.756)*	88 *(σ=3.250)*

	1-500	IMGP 10-50	PADGP 10-50
Even Parity 4	56 *(σ=4.964)*	50 *(σ=5.000)*	77 *(σ=4.208)*

	1-250	IMGP 5-50	PADGP 5-50
Symbolic Reg.	44 *(σ=4.964)*	53 *(σ=4.990)*	56 *(σ=4.964)*

Fig. 3.7. Summary of the behavior of GP (first data column), IMGP (second column), and communicating-island GP (third column) for the test problems for some selected population sizes. The number of successes and their standard deviations are shown

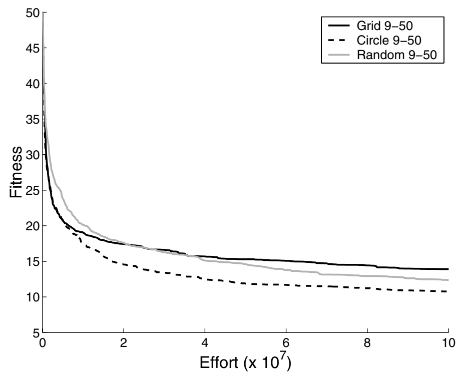
The graph of fitness against effort shows that the ring and random topologies achieve better results than does the grid topology. In any case, however, both the number-of-hits table and the histograms of the relative frequency of solutions show that the differences between results obtained with the various topologies used are marginal, a fact that is also confirmed by the standard deviations on the success rates.

Figure 3.9, showing results for 16 populations of 100 individuals each, confirms this trend, with the curves of the circle and random topologies being even closer, while the grid topology shows the worst performance.

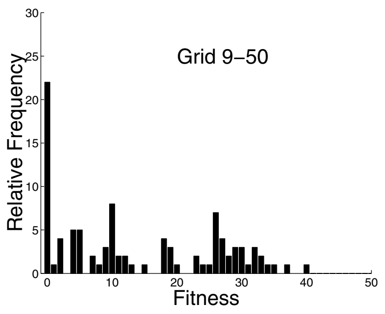
Figure 3.10 depicts results for the even-parity-4 problem in the case of nine populations of 50 individuals.

Here the grid topology gives the best results, in agreement with the findings of Andre and Koza [11], although the influence of communication topology on the evolutionary process appears to be marginal.

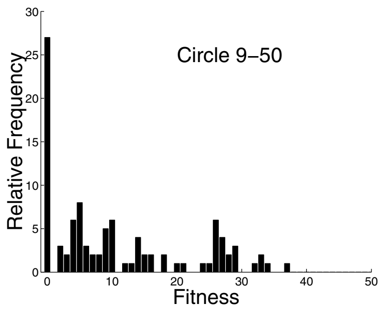
In conclusion, at least for the problems studied here, the differences between the results obtained with the various topologies are narrow, and thus topology does not seem to be the most important factor in multipopulation GP. On the other hand, the random and ring topologies possess an advantage in terms of communication efficiency. In fact, an s-subpopulation ring or random system sends only s messages at each iteration, whereas an $n \times n = s$ grid topology sends $4 \times s$ messages, all of the same size. When all this considered, using a random topology seems to be advisable in view of the above results, given that this topology is easy to implement and seems more natural, as it does not prescribe a fixed exchange pattern.



(a)



(c)



(d)

	$E = 6\times10^7$	$E = 8\times10^7$	$E = 10\times10^7$
Grid 9-50	18	20	22 (σ=4.142)
Circle 9-50	26	27	27 (σ=4.439)
Random 9-50	19	22	24 (σ=4.270)

(b)

Fig. 3.8. The artificial ant problem. Island GP with three different communication topologies. (a) Average fitness against effort for 9 populations of 50 individuals each over 100 executions. (b) Number of successes in 100 runs for three values of the total effort. (c) Histogram of the relative frequency of the solutions found with a grid communication topology for an effort $E = 10 \times 10^7$. (d) Histogram of the relative frequency of the solutions for a ring topology for the same value of the effort

These results are not surprising. After all, multipopulation EAs represent only a relatively small change with respect to a panmictic population: most of the time, the system evolves as if it were made of a number of independent mixing populations. The periodic “perturbations” caused by migrants leaving and entering populations do not fundamentally change the picture.

3.7 Migration Parameters

Up to now, it has been implicitly assumed that the values of the number of migrants (10% of the island size) and of the frequency of migration (every ten generations) were suitable. But these are two free parameters of the system, and they could have been chosen differently. It is true that these parameters

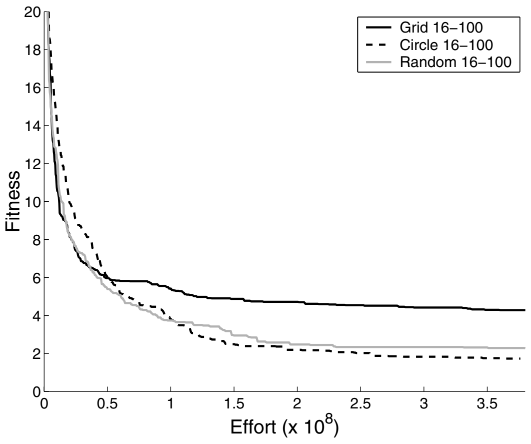


Fig. 3.9. The artificial ant problem. Island GP with three different communication topologies. Average fitness against effort for 16 populations of 100 individuals each, over 100 executions

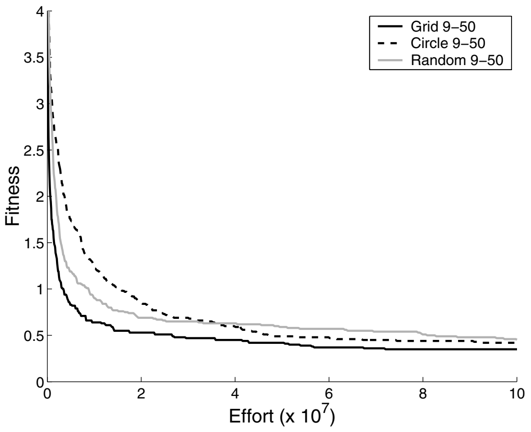


Fig. 3.10. The even-parity-4 Problem. Island GP with three different communication topologies. Average fitness against effort for 9 populations of 50 individuals each, over 100 executions

were chosen not at random, but rather by an examination of values previously used with success. It is nevertheless advisable to get at least a feeling for their influence on the evolutionary process in order to be able to make an educated guess about this influence.

Let us call the number of individuals that are exchanged between islands the *grain*, and the number of generations that elapses between two successive exchanges of individuals between subpopulations the *period*. The graphs shown in Figs. 3.11 and 3.12 illustrate the fitness level reached as the period and the grain are varied, after fixing a maximum effort of computation for each problem. This threshold was fixed at the value of the effort reached after 500 generations. The experiments were run with five populations of 100 individuals each, for the artificial ant and even-parity problems. The curves are averages of 100 independent executions of the same experiment.

Even-Parity-4 Problem

Results for this problem are shown in Fig. 3.11, where fitness curves are given as a function of the grain for a number of values of the period. This figure shows that the best value of the fitness is reached by the curve representing a period of 10, for a value of the grain equal to 10. We also note that for low grain values it is preferable to exchange individuals at each iteration (i.e. period = 1). For values of the grain from 5 to 20 it is better to exchange individuals every 10 iterations, but a value of the period equal to 5 also gives satisfactory results. For values of the grain greater than 25, it is better to exchange individuals less frequently, i.e. each 20 or 25 iterations. This was expected, since too much mixing of the populations slows down the convergence process.

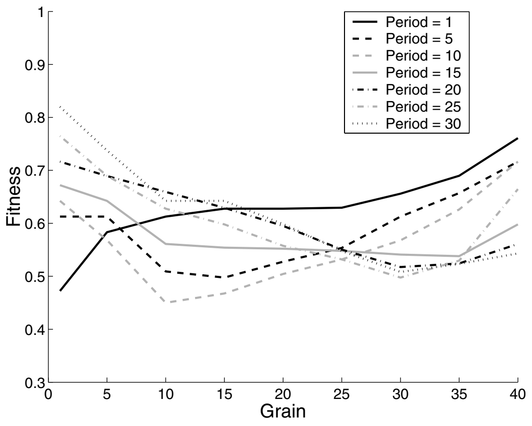


Fig. 3.11. The even-parity-4 problem. Fitness as a function of the grain for several values of the period. Five subpopulations of 100 individuals each, with random communication topology

Artificial Ant Problem

The averaged results for the ant problem are depicted in Fig. 3.12. This figure shows that the best value of the fitness is reached by the curves representing periods of 5 and 10, for a value of the grain equal to 10. We also notice that for low values of the grain, it is in general better to exchange individuals at each iteration (i.e. period = 1). For values of the grain from 5 to 25 it is better to exchange individuals with a period of 1, 5, or 10 iterations. For values of the grain greater than 30, exchanging individuals less frequently gives better results owing to a smaller mixing effect in the subpopulations. However, for the ant problem, large values of the grain always give worse results.

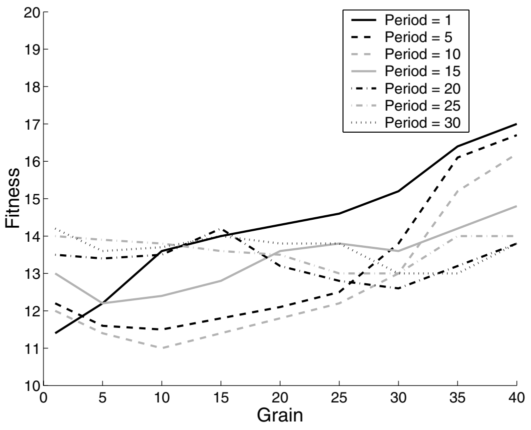


Fig. 3.12. The artificial ant problem. Fitness as a function of the grain for several values of the period. Five subpopulations of 100 individuals each, with random communication topology

Another work on GP that deals with the influence of the number of individuals exchanged is [11]. The authors of [11] studied the effort of computation as a function of the grain and found that a grain of 5% of the population worked best. Other values around 4–8% are also good, thus confirming qualitatively the present study. The communication topology used in [11] was two-dimensional grid instead of a random topology, and the authors of that work did not study the frequency of migration, since their system used an asynchronous message-passing pattern.

The results obtained for migration parameters confirm that, on the whole, the ranges of values that have been empirically chosen over the years are reasonable. For best results, the number of individuals to be sent to another population should be about 10% of the population size, and the exchanges should take place every five to ten generations. The results also make it clear that if few individuals are exchanged, then it is best for this to be done frequently. In contrast, a large grain should go hand in hand with a low frequency of exchange, otherwise the genetic material in the populations does not have time to improve sufficiently from one exchange to the next. But too large a grain slows down the convergence process, since this brings about a homogenization effect that counters the necessary genetic drift of the artificial evolution. All things considered, frequent exchanges of a few individuals are to be preferred to infrequent migrations of large blocks.

3.8 Summary of Case Study

This case study sheds some light on the behavior of multipopulation GP. Two fundamental aspects are the overall size of the working populations and the number of islands. One can see empirically that, as long as the size of the subpopulations is above a minimum threshold, a distributed system is more effective than a single panmictic population of the same total size. This is true even for the case of isolated islands, but the best results are obtained when a certain amount of migration is allowed. Obviously, the useful range of population size is problem-dependent and has to be determined empirically (but see the previous chapter for some attempts at sizing the populations based on theoretical principles).

Other, finer aspects of island EAs have also been studied. Concerning the migration topology, the broad conclusion is that the communication architecture does not have a marked influence on the results. However, the random topology is at least as good as the ring and the grid, two other widely used graphs. The number of individuals that migrate and the frequency of migration are consistently similar across problems, and confirm the range of values used in previous work.

The considerations above are based on a small number of problems. However, many other cases showing similar behavior are documented in the literature (see, for instance, [5] and references therein), especially for island GAs, giving strength to the empirical conclusions.

Several reasons have been invoked in the literature to justify the undeniable practical effectiveness of island EAs. Preservation of diversity is one benefit that is commonly attributed to island and other structured EAs. Another suggested advantage is the possibility of enhancing the explorative characteristics of the EA owing to the semi-isolation of the populations. Actually, although these capabilities are potentially available, whether or not they are put to good use will depend on the type of problem and on the choice of parameters. To help us get a feel for one of these factors, the next section looks into the issue of diversity in multipopulation GP.

3.9 The Role of Diversity

Diversity among the individuals in a population plays an important role in EAs. One of the shortcomings of standard EAs is their inability to maintain diversity in the population. This lack of diversity can lead to a number of problems, such as converging to nonglobal optima and not being able to react to changes in the environment. Poor diversity is especially troublesome when one is dealing with multimodal problems or using EAs to solve dynamical problems.

In genetic programming, the process converges when the elements of the phenotypic pool are identical or nearly so, in spite of the fact that the genotypic pool might still show some syntactical diversity. When this occurs, the crossover operator ceases to produce new good individuals, and the algorithm allocates all of its trials in a very small subset of the program space. Unfortunately, this often occurs before the true optimum has been found; this behavior is called premature convergence. The mutation operator provides a mechanism for reintroducing lost diversity, but it does so at the cost of slowing down the learning process.

Both genotypic and phenotypic diversity play a role in GP, and the two are not necessarily correlated in a straightforward manner [23]. In particular, the phenomenon of "bloat", the tendency of GP code to grow in size with generations is well known, and it often gives rise to large nonfunctional portions of the tree that could increase genotypic but not phenotypic diversity, and cannot increase the capability of the system to produce better solutions either.

Several "explicit" approaches have been proposed for maintenance of diversity within a population. Of these, fitness sharing is the oldest [40], while multiobjective optimization methods [38] try to preserve diversity by considering fitness, size, and diversity as simultaneous objectives to be satisfied. While these methodologies are worthwhile in their own right, here we shall examine only "implicit" diversity maintenance through the spatial structure of the population.

3.9.1 Diversity Measures

A good survey and discussion of diversity measures in panmictic GP has been presented in [23]. Other EA families will need different measures, of course, but the concepts are the same. For example, in binary-coded GAs, the Hamming distance is often used. The measures used here are based on the concepts of *entropy* and *variance*, and are employed to evaluate the *phenotypic* (i.e. based on fitness) and *genotypic* (i.e. based on the syntactical structure of individuals) diversity of populations. Phenotypic diversity is related to the number of different fitness values of the individuals. The corresponding *phenotypic entropy* $H_p(P)$ [121] of a population P is

$$H_p(P) = -\sum_{j=1}^{N} f_j \log(f_j), \tag{3.1}$$

where f_j is the fraction n_j/N of individuals in P having fitness j, and N is the number of fitness values in P.

Genotypic entropy can be used to measure genotypic diversity. However, GP individuals are usually represented as variable-size trees, and a tree distance measure is needed to define structural differences between trees. One useful definition of tree distance has been proposed by Ekárt and Németh [48]. The distance between two trees T_1 and T_2 is calculated in three steps. (1) T_1 and T_2 are overlapped at the root node and the process is applied recursively, starting from the leftmost subtrees. (2) For each pair of nodes at matching positions, the difference between their codes (possibly raised to an exponent) is computed. (3) The differences computed in the previous step are combined into a weighted sum. Formally, the distance between two trees T_1 and T_2 with roots R_1 and R_2 is defined as follows:

$$dist(T_1, T_2) = d(R_1, R_2) + \frac{1}{k}\sum_{i=1}^{m} dist(child_i(R_1), child_i(R_2)), \tag{3.2}$$

where $d(R_1, R_2) = (|c(R_1) - c(R_2)|)^z$, $child_i(Y)$ is the ith of the m possible children of a generic node Y, if $i \leq m$, or the empty tree otherwise; and c, evaluated on the root of an empty tree, is 0. The constant k is used to give different weights to nodes belonging to different levels, and z is a constant usually chosen in such a way that $z \in \mathcal{N}$.

With this definition of tree distance at hand, we can now define the genotypic entropy $H_g(P)$ of a population P as follows:

$$H_g(P) = -\sum_{j=1}^{N} g_j \log(g_j), \tag{3.3}$$

where g_j is the fraction of individuals having a given distance from the origin, which has been chosen arbitrarily as the empty tree.

The variance gives an alternative metric for diversity. The variance of a population P is defined as follows:

$$V(P) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2. \tag{3.4}$$

If we are considering phenotypic variance, $\overline{x}$ is the average fitness $\overline{f}$ of the individuals in P, and x_i is the fitness f_i of the ith individual in P. N is the total number of individuals in P. The notion of tree distance is used to define genotypic variance. In this case, $\overline{x}$ is the average of all the individual distances from the origin tree, and x_i is the distance of the ith individual in P from the origin tree.

3.9.2 Experimental Results

This section describes the results related to diversity for the simulations of the GP test problems described earlier. All the curves represent average values over 100 independent GP runs.

Artificial Ant Problem

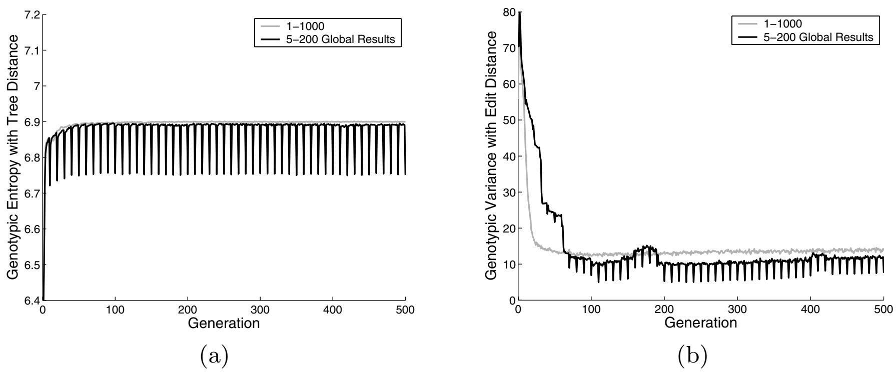


Fig. 3.13. Artificial ant problem. 1000 total individuals. Thec curves are averages over 100 runs. (a) Genotypic entropy using structural distance. (b) Genotypic variance using structural distance. Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

Figure 3.13 a depicts the behavior of the genotypic entropy calculated by using structural tree distances. The gray curve represents the entropy of one panmictic population, while the black curve shows the aggregated entropy of all islands, i.e. the entropy of all the individuals in the islands considered as a single population. Figure 3.13 b shows the genotypic variance for each generation. We observe that genotypic diversity, after an initial reorganization (an increase in the the case of entropy, and a decrease in the case of the variance), tends to remain constant over time. This is in agreement with the findings in [23]. The jagged behavior of the multipopulation curves when groups of individuals are sent and received is not surprising: it is due to the sudden change in diversity when new individuals enter a subpopulation. If we do not consider these oscillations, the genotypic diversity of the panmictic population and that of the aggregated subpopulations can be considered to be very similar.

The behavior of the genotypic diversity in individual islands can be seen in Fig. 3.14, where only two populations are reported to avoid cluttering the drawing. The general behavior in the islands is similar to that of a single larger population. However, two effects are visible: first, the diversity of the islands is lower and, second, the diversity fluctuations are larger. Both phenomena are easily understandable if we remember that the population size is smaller in the individual islands.

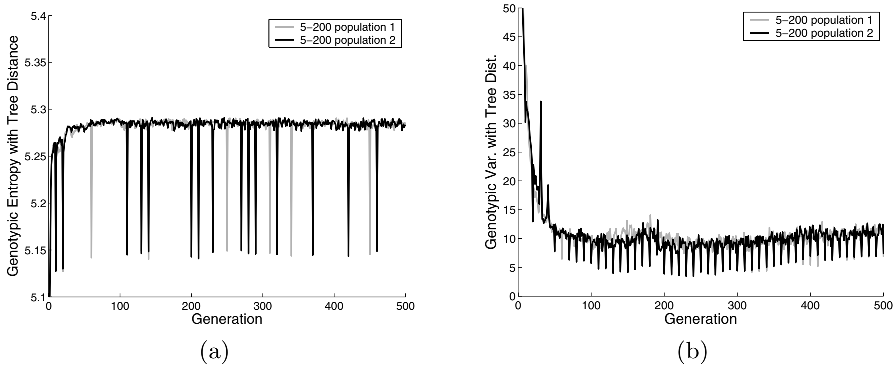


Fig. 3.14. Artificial ant problem. Genotypic entropy (a) and variance (b) using structural distances, in two subpopulations of 200 individuals each

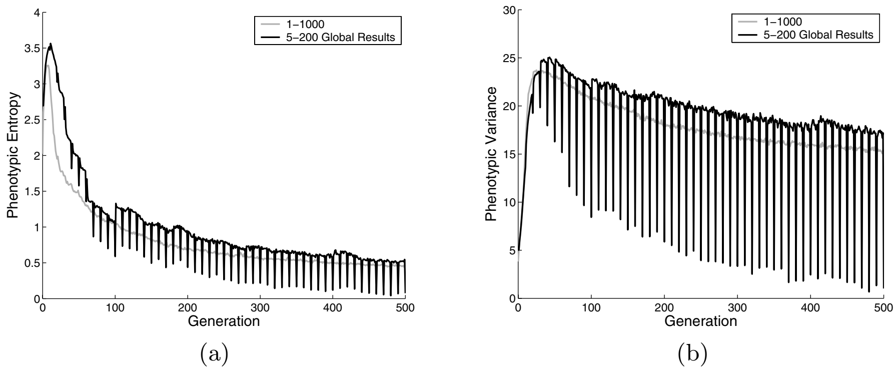


Fig. 3.15. Artificial ant problem. 1000 total individuals. Phenotypic entropy (a) and variance (b). Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

Figure 3.15 shows graphs of the phenotypic entropy and variance for both the panmictic population and the multipopulation case. It is apparent here that, unlike the genotypic diversity, the phenotypic diversity tends to decrease steadily with time, which is in agreement with the results of [23] for panmictic GP. The same behavior has often been observed in GP runs; see, for instance, [92]. The interesting remark that we can make is that, even though the average phenotypic diversity tends to oscillate in the multipopulation case as groups of individuals are sent and received, it remains higher globally than in the panmictic case.

Symbolic Regression Problem

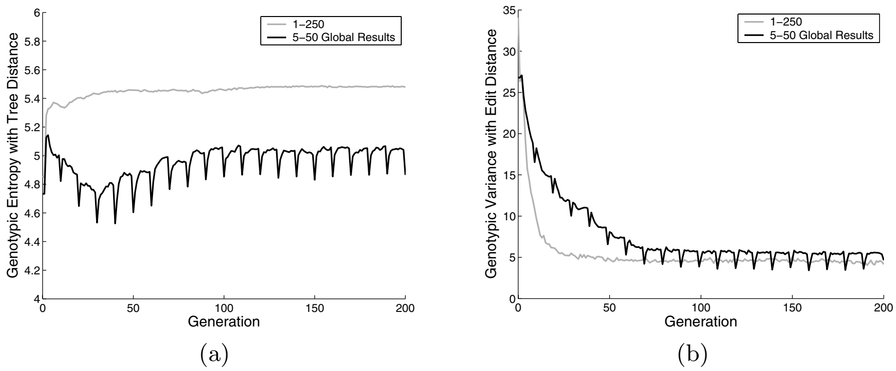


Fig. 3.16. Symbolic regression problem. 250 total individuals. Genotypic entropy (a) and variance (b) calculated using structural tree distance. Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

The genotypic entropy and variance obtained with the structural tree distance are shown in Fig. 3.16. We see that, after an initial transient period, the genotypic diversity remains approximately constant during the evolution. Moreover, while the genotypic variance has more or less the same values for the multipopulation and the panmictic systems, the genotypic entropy is higher for the single panmictic population. The smaller values of genotypic entropy in the multiisland system do not seem to affect performance, as we saw in Sect. 3.5.2.

Figure 3.17 shows the phenotypic diversity (as measured by entropy and variance) for the multipopulation and the panmictic systems. It appears that, on average, the phenotypic diversity is higher in the multiisland case. This is a qualitative confirmation that distribution helps in maintaining phenotypic diversity. An inspection of Figs. 3.16 and 3.17 suggests that there is little correlation between genotypic and phenotypic diversity (which is in agreement with [23], and is probably due to bloat, neutral networks in genotypic space, and nonfunctional code [92]). Moreover, no apparent correlation seems to exist between the ability of GP to find good-quality solutions and the genotypic diversity. On the other hand, the capacity of GP to find good-quality solutions seems to be correlated with the phenotypic diversity, at least in these experiments.

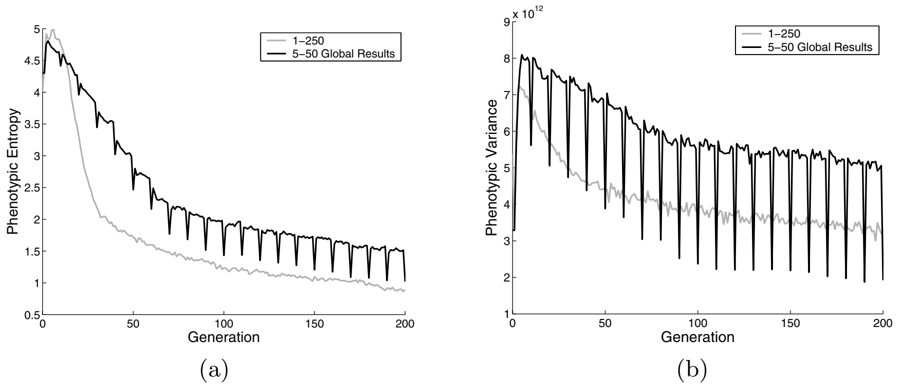


Fig. 3.17. Symbolic regression problem. 250 total individuals. Phenotypic entropy (a) and variance (b). Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

Even-Parity-4 Problem

In Fig. 3.18, we observe that a pattern of genotypic diversity similar to that found for the artificial ant problem (Fig. 3.13) emerges again: the genotypic diversity changes in the initial part of the evolution: the entropy increases and the variance decreases, and then levels off and stays practically constant. For the islands, there are oscillations at migration times, but otherwise the behavior is quite similar to that of the panmictic system.

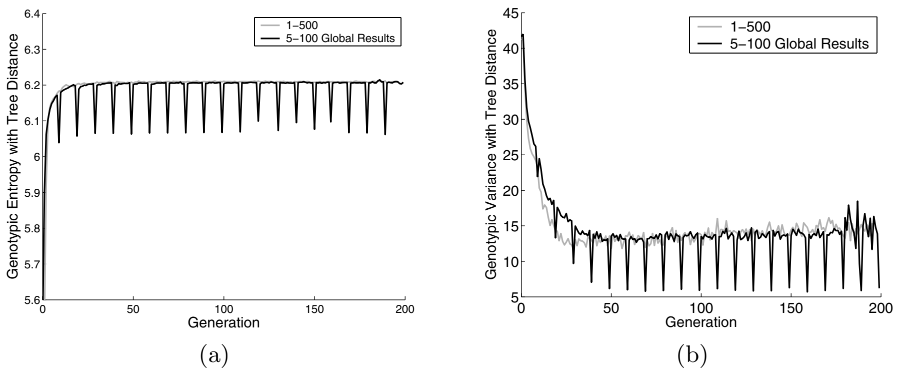


Fig. 3.18. Even-parity-4 problem. 500 total individuals. Genotypic entropy (a) and variance (b) calculated using the structural distance. Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

Also, for this problem, the phenotypic diversity (the entropy and variance are shown in Fig. 3.19) always decreases on average during evolution, but it remains higher for the multipopulation system.

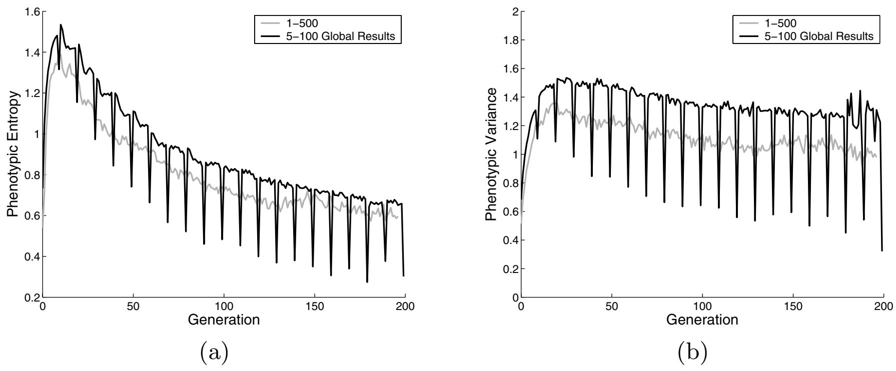


Fig. 3.19. Even-parity-4 problem. 500 total individuals. Phenotypic entropy (a) and variance (b). Gray curves: panmictic population. Black curves: entropy of the aggregated subpopulations

3.9.3 Summary

We have seen how using loosely coupled populations instead of a single panmictic one may help in maintaining diversity during GP runs. By defining indices of genotypic and phenotypic diversity and by monitoring their variation over a large number of runs in three standard test problems, it has been shown experimentally that diversity evolves differently in the multipopulation case. In fact, while genotypic diversity is not much affected by splitting a single population into multiple ones, phenotypic diversity, which is linked to fitness, remains higher in the multipopulation case for all problems studied here. Thus, the low correlation between the evolution of genotypes and problem-solving behavior is confirmed in the distributed setting [23]. Given that better convergence properties have been empirically established for distributed GP in this chapter, it is tempting to attribute this observation to the behavior of the diversity. Although the data suggest that this could indeed be the case, maybe through an implicit control of the bloat phenomenon (as suggested in [60]), a direct effect cannot be established, only a plausible indication.

In conclusion, using multiple loosely coupled populations is a natural and easy way to maintain diversity and, to some extent, avoid premature convergence in GP. Of course, more complicated methods for promoting diversity (e.g. [38, 48]) may be used in the subpopulations in conjunction with the natural distribution of the genetic material offered by the latter.

3.10 Asynchronous Island Models

This last section deals with the empirical effect of asynchronous communications in island-based GP. We have already seen general descriptions of

asynchronous island models in Sect. 2.4. Here, some experimental results are reported on the benchmark problems of this chapter for two asynchronous multipopulation GP models, and are compared with the synchronous island model and with standard GP (more details can be found in [149]).
Two asynchronous island models were used for these experiments: a model with a master process that coordinates the individual exchanges, and a model without a master process. In these models, blocks of migrants are sent synchronously, and are received asynchronously in the destination islands. As we have seen in Chap. 2, both the sending and receiving could be asynchronous instead. The synchronous model that the results are compared with is a standard one, except that, as in the first asynchronous algorithm, it has a special master process that takes care of receiving blocks of individuals and dispatching them to the target population. This arrangement containing a master process is irrelevant from the point of view of the abstract model, except maybe for fault-tolerance considerations, but it makes it easier to implement any island interconnection topology.

Asynchronous Communication with Master

The behavior of the populations can be described by the following algorithm:

```
• Create a random population of programs
• **While** termination condition not reached **do**
    – Assign a fitness value to each individual
    – Select a set of individuals for reproduction
    – Recombine and mutate the new population
    – **If** communication has to take place at this iteration **then**
        · Select the best $n$ individuals and send them to the master
        · Receive a set of $n$ new individuals from the master
      **EndIf**
    – Test the completion of all the pending receives
    – **For** all the terminated receives
        · replace the $n$ worst individuals in the population with the $n$ received
          individuals
      **EndFor**
  **EndWhile**
```

At the same time. the master executes the following steps:

```
• **For** each iteration in which communication has to take place **do**
    – **For** each population $p$ **do**
        · Receive a set of $n$ individuals from $p$
        · Send them to another population according to the chosen topology
      **EndFor**
  **EndFor**
```

Asynchronous Communication Without Master

In this model each population executes the following steps:

```
• Create a random population of programs
• While termination condition not reached do
    – Assign a fitness value to each individual
    – Select a set of individuals for reproduction
    – Recombine and mutate the new population;
    – If communication has to take place at this iteration then
        · Select the best $n$ individuals and send them to another process according
          to the chosen topology
        · Receive a set of $n$ new individuals from another process according to the
          chosen topology with a nonblocking receive operation
      EndIf
    – Test the completion of all the pending receives
    – For all the terminated receives
        · replace the $n$ worst individuals in the population with the $n$ received
          individuals
      EndFor
  EndWhile
```

GP Parameters

In all the experiments performed, the same set of GP parameters was used: generational GP, crossover rate 95%, mutation rate 0.1%, tournament selection of size 10, ramped half-and-half initialization, maximum depth of individuals for the creation phase 6, maximum depth of individuals for crossover 17, no elitism. The subpopulations were connected using a ring topology, and a number of individuals equal to the 10% of the size of the subpopulation were exchanged between demes every 10 generations.

The systems were compared with respect to the computational effort spent to reach a given average solution quality, and the phenotypic entropy, as defined in Sect. 3.9, was used to gauge the diversity in the populations.

3.10.1 Experimental Results

Fitness vs. Computational Effort

Owing to the stochastic nature of the evolutionary process, all the results shown here were obtained by averaging 60 independent runs of the same experiments. Curves are drawn for the three distributed models and for the reference standard GP model. In order to assess the statistical significance of the results, standard deviations of the mean fitness are also presented. Figures 3.20, 3.21, and 3.22 show graphs of fitness and standard deviation versus computational effort for the even-parity-5 problem, the artificial ant problem, and the symbolic regression problem, respectively. The curves for the symbolic

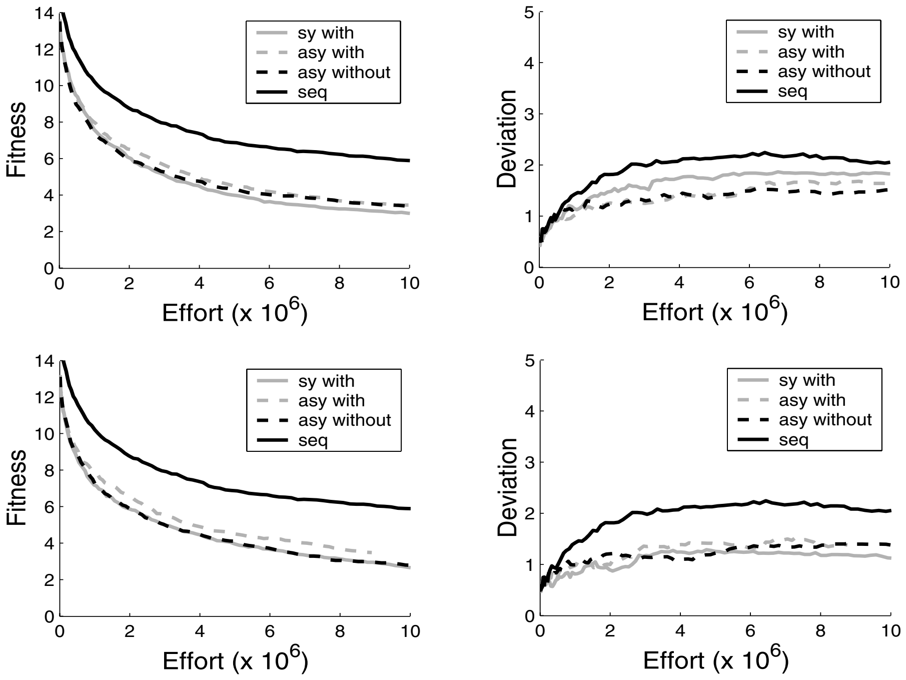


Fig. 3.20. Fitness and its standard deviation against computational effort for the even-parity problem. Upper graphs, 5 populations of 300 individuals, lower graphs, 10 populations of 150 individuals. Sequential version (a panmictic population of 1500 individuals) shown in black

regression problem look like straight lines because of the fast convergence of this easy problem. From these figures, we observe that the distances between the curves for the distributed models and those for the sequential versions are always larger than the standard deviations, while this is not the case if we compare the curves of the parallel models between each other. One can thus conclude that, for all the cases presented, the island models have a faster convergence than the panmictic model, while the speeds of convergence of the synchronous and asynchronous models can be considered statistically equivalent.

Population Phenotypic Entropy

The results for this quantity were obtained by averaging 60 independent runs of the same experiments. The curves relating to multipopulation models were obtained by averaging the phenotypic entropies of all the subpopulations at each iteration. Figures 3.23, 3.24 and 3.25 show the population entropy versus the generation number for the even-parity-5 problem, the artificial ant problem, and the symbolic regression problem, respectively. These figures show

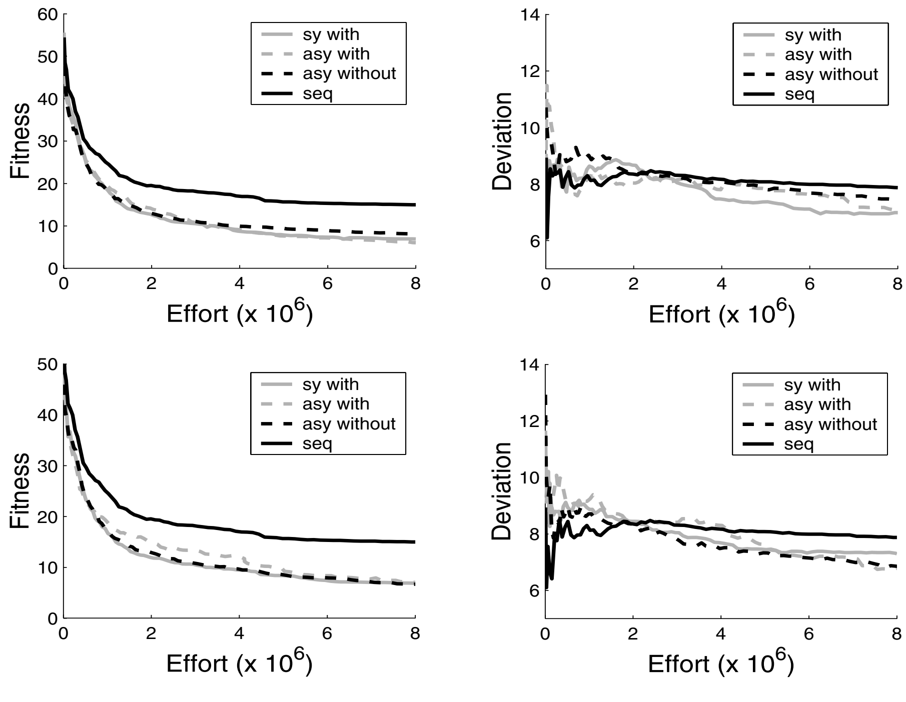


Fig. 3.21. Fitness and its standard deviation against effort for the ant problem. Upper graphs, 5 populations of 300 individuals, lower graphs, 10 populations of 150 individuals. Sequential version (a panmictic population of 1500 individuals) shown in black

that the panmictic model has a fast phenotypic entropy increase in the first few generations and a slow decrease in the rest of the execution. The synchronous model shows a sudden decrease in the entropy each time the generation number is a multiple of 10 (messages are sent synchronously every ten generations), and a sudden increase in the immediately following few generations. This behavior, due to the sudden arrival of new good individuals, was found earlier in Sect. 3.9.2.

The asynchronous models, on the other hand, show an oscillating behavior, due to the fact that messages are received whenever they arrive, and not at fixed times. In all cases, migrating individuals seems to help to alleviate stagnation in the populations, thus confirming one of the intuitions that are the basis of distributed evolutionary algorithms.

To summarize the findings of this section, one can say that asynchronous operation does not cause a marked difference in the behavior of the island model. Except during the few generations after the reception of new individuals in an island, the average behavior is roughly the same for synchronous and asynchronous multipopulation models. On the other hand, the advantage of

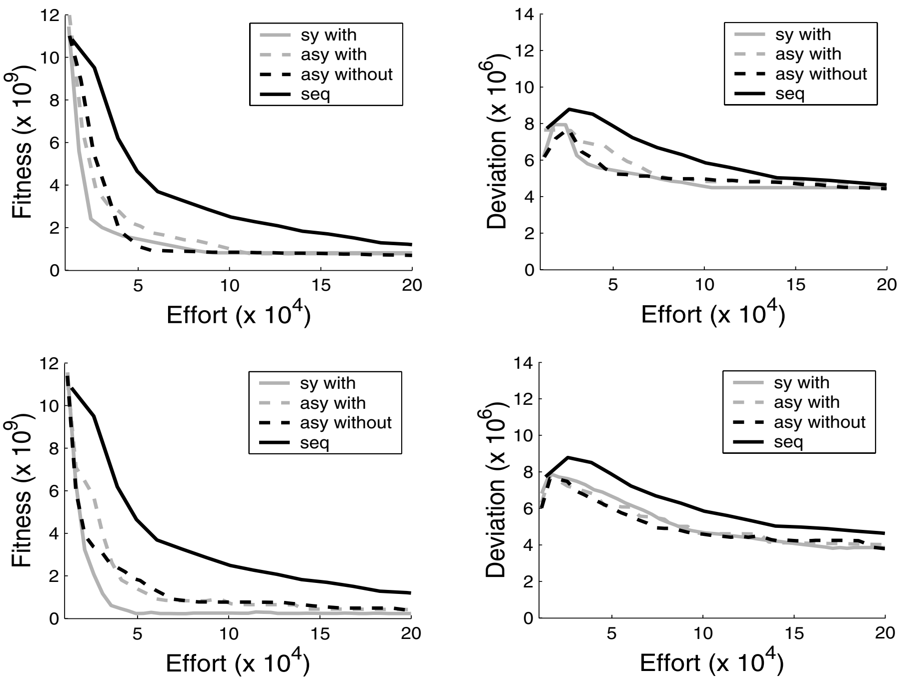


Fig. 3.22. Fitness and its standard deviation against effort for the symbolic regression problem. Upper graphs, 5 populations of 300 individuals, lower graphs, 10 populations of 150 individuals. Sequential version (a panmictic population of 1500 individuals) shown in black

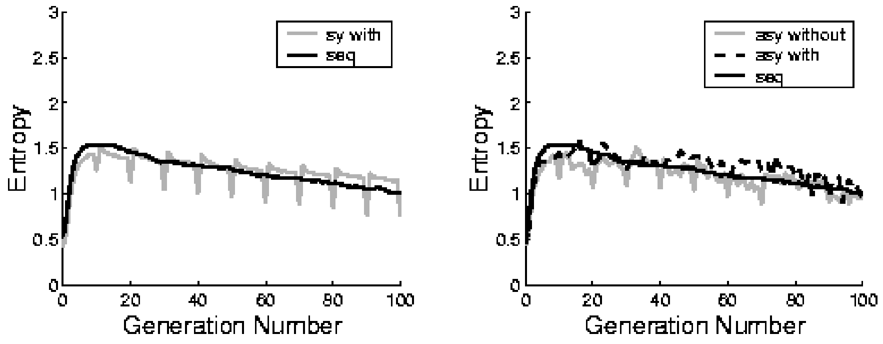


Fig. 3.23. Phenotypic entropy against generation number for the even-parity-5 problem (5 populations of 300 individuals each). Curves for the sequential version (a panmictic population of 1500 individuals) are shown too

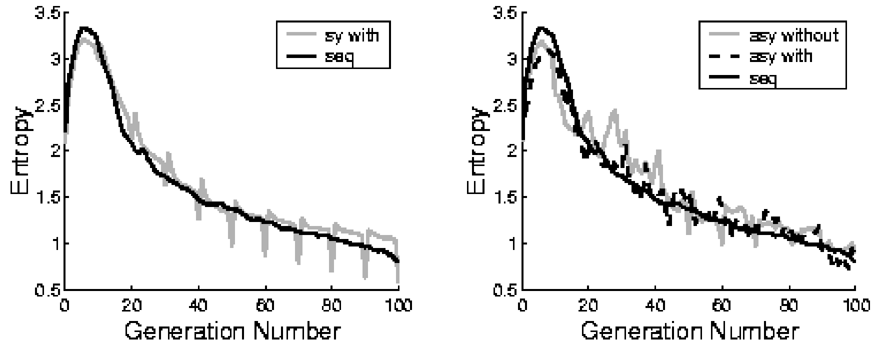


Fig. 3.24. Phenotypic entropy against generation number for the artificial ant problem (5 populations of 300 individuals each). Curves for the sequential version (a panmictic population of 1500 individuals) are shown too

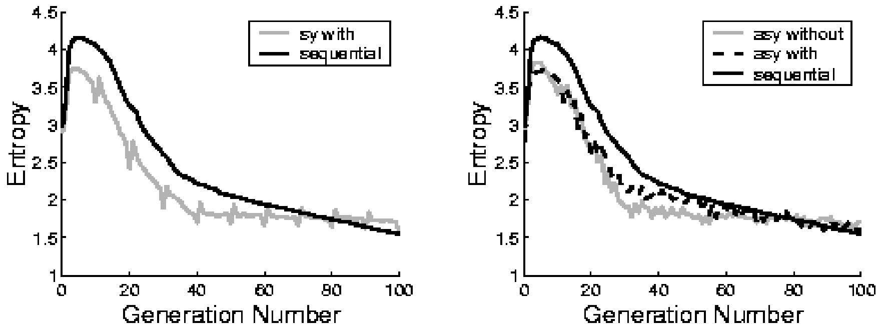


Fig. 3.25. Population phenotypic entropy against generation number for the symbolic regression problem (5 populations of 300 individuals each). Curves for the sequential version (a panmictic population of 1500 individuals) are shown too

those models over the panmictic model is once again confirmed with respect to computational effort, the quality of the solutions found, and overall diversity.

At this point, since we have been using the same few test functions throughout this chapter, the reader might be led to believe that the empirical evidence presented in favor of the island model might be limited to those particular problems and GP. To indicate that this is not the case, it is perhaps useful to give pointers to a few other experimental investigations with island models, in addition to those already mentioned in Chap. 2.

Starkweather et al. [143], using a steady-state replacement scheme, showed that for most of their test functions and for most experimental conditions, a ring island topology outperformed a panmictic population of the same total size. These authors used a migration scheme that swapped migrants between populations on the ring first between first-neighbor islands, then between second-neighbors, and so on modulo the ring size.

Andre and Koza [11] used a multipopulation GP system in which the interconnection topology was a two-dimensional grid. The send and receive operations for the migrating individuals were fully asynchronous. Andre and Kota found that the algorithm was more efficient than a panmictic one on boolean even-parity functions.

Alba and Troya performed a rather systematic study of structured GAs, including multipopulation models and their convergence and population diversity behavior [8]. Using a completely different set of test functions suited to GA encoding, they showed that both synchronous and asynchronous islands outperformed the panmictic model. In addition, they found that, in a physically distributed environment, both showed excellent speedup, with the asynchronous version being more efficient than the synchronous one owing to a lower communication overhead.

Many other studies not reported here, including real-life applications, tend to lead to the same conclusions. In fairness, it should be mentioned that, although there seems to be a general consensus about the advantages offered by multipopulation EAs, some dissenting voices have been raised for example, Punch's work on multipopulation GP [119]. However, Fernández et al. have argued in [51] that this apparent discrepancy may be due to insufficient significance of the statistics.

Finally, I would like to remind the reader that any EA can be improved by combining it with a problem-specific local search heuristic or with tailored operators, giving rise to what is known as a *hybrid* EA. Of course, the same can be done in the case of island GAs, although I shall not present the corresponding kind of algorithm explicitly here. An early example of an island hybrid EA in the field of function optimization is the work of Mühlenbein et al. [108], in which a local optimizer is used.

4 Lattice Cellular Models

Lattice-structured populations were defined in graph-theoretic terms in Sect. 1.1. This kind of spatially structured EAs was introduced early on in EA research by Gorges-Schleuter, Manderick, and coworkers [70, 96] and has been used quite often since then. Thanks to the geographical isolation of the individuals in the population, these populations feature slow diffusion of good solutions through the lattice, and thus, for a given selection method, their evolution leads to a more explorative behavior than panmictic EAs. The reasons for this behavior will become clear in this chapter. These aspects have been found useful for multimodal and other kinds of problems, as we shall see in the next chapter, where I describe the application of cellular EAs (cEAs) to a set of test problems.

Lattice cellular EAs represent a more radical departure from the panmictic population scheme than do the island models that we have studied earlier. They are thus more useful in understanding the properties of populations in which locality is a key factor. In this chapter, we turn our attention to the dynamical properties of these structured populations. As an important case study, I shall present models for the global, emerging *selection pressure* in cEAs. Selection is an extremely important operation in artificial evolution: it is the process of choosing individuals for reproduction or survival and it thus provides the driving force in EAs. Another advantage of studying selection in isolation stems from the difficulties in mathematically modeling the interplay of selection and variation operators. Our goal is to explain how population topology influences the behavior of an EA, and this is easier to see when different effects are first treated separately.

In this chapter we shall also study in depth the influence of the timing of events on the evolutionary process, a subject that we have encountered already in connection with island models (see Chap. 2). The influence of asynchronous operations was seen to be relatively minor there. With cellular populations, however, we shall see that time matters, and the interplay of time and topological structure is a very interesting one, with rather important consequences for the dynamics of evolution.

The following section gives a brief description of the concept of selection pressure, or *selection intensity*[1], and the related idea of *takeover time.*

4.1 Takeover Time

Selection methods are characterized by their takeover time. The *takeover time* is the time it takes for a single, best individual to take over the entire population. In other words, it represents the speed at which the best solution in the initial population propagates and conquers the whole population under the application of the selection operator alone. It can also be seen as a simplified infective process, in which infection means being replaced by the best individual and in which infected individuals remain infected forever. The takeover time can be estimated experimentally by measuring the proportion of the best individual as a function of time, under the effect of selection only, without any variation operator. A shorter takeover time indicates a higher selection pressure, and thus a more exploitative algorithm. If the selection intensity is lowered, the algorithm becomes more explorative. Selection pressure is thus a key parameter in the operation of an EA: with high selection pressure, diversity in the population is quickly lost, and the search stagnates, unless a large population is used or a lot of disruption is caused by the variation operators. On the other hand, if the selection pressure is weak, convergence slows down and the search may wander in the problem space without focusing on very good solutions. An effective search thus requires a careful trade-off between the selection method, the variation operators, and other EA parameters such as the population size. We shall see some of these effects in the empirical studies described in the next chapter.

Theoretical takeover times have been derived by Goldberg and Deb [67] and by Bäck [14] for panmictic populations and the selection methods used across the families of EAs. These times turn out to be logarithmic in the population size, except in the case of proportional selection, which is a factor of n slower, where n is the population size.

It has been shown empirically in [126] that the selection pressure induced on the entire population becomes weaker when we move from a panmictic to a square-grid population of the same size, with synchronous updating of the cells. A theoretical study of the selection pressure in the case of ring and array topologies in one-dimensional cEAs has been done by Rudolph [124]. Abstracting from specific selection methods, he splits the selection procedure into two stages: in the first stage an individual is chosen in the neighborhood of each individual, and then, in the second stage, for each individual it is decided whether the other chosen individual will replace it in the next time step. Using

[1] Note that we use the term *selection intensity* here rather informally, to express the degree of explorative or exploitative character of the algorithm. Selection intensity is a rigorous concept in population genetics [107].

only replacement methods in which extinction of the best by chance cannot happen, i.e. nonextinctive selection, Rudolph derived the expected takeover times for the two topologies as a function of the population size and the probability that, in the selection step, the individual with the best fitness is selected in the neighborhood.

In the rest of this chapter, I shall complete and extend the theoretical results on selection pressure in lattice-structured cellular EAs, including both synchronous and asynchronous cell update modes, for one- and two-dimensional grids.

4.2 Synchronous cEAs

We begin with a description of how a standard synchronous cEA works. A lattice cEA maintains a population whose individuals are spatially distributed in cells according to a regular lattice structure such as the grid shown in Fig. 4.1. Each cell is occupied by one individual; therefore, the terms *cell* and *individual* may be used interchangeably.

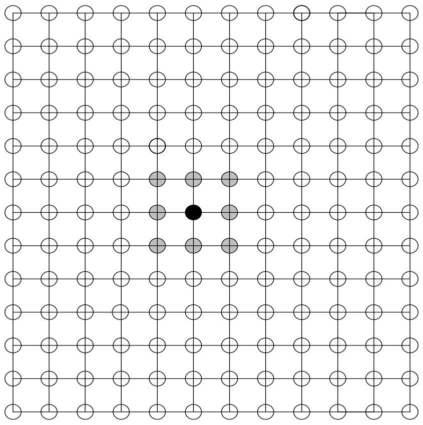


Fig. 4.1. A grid cellular structure with a Moore neighborhood highlighted in gray around the central black cell

A cEA starts with the cells in a random state and proceeds by successively updating them using evolutionary operators, until a termination condition is satisfied. Updating a cell in a cellular EA means selecting parents in the individual's neighborhood, applying genetic operators to them, and finally replacing the individual if the offspring obtained has a better fitness (other replacement policies can be used). The following is a high-level pseudocode description of an algorithm for a general cEA:

```
for each cell $i$ in the grid do in parallel
    generate a random individual $i$
end parallel for
while not termination condition do
    for each cell $i$ in the grid do in parallel
        Evaluate individual $i$
        Select individual(s) in the neighborhood $K(i)$
        Produce offspring
        Evaluate offspring
        Assign one of the offspring to cell $i$ according to some criterion
    end parallel for
end while
```

This chema of an algorithm can be specialized for different types of EAs. For instance, in cellular genetic algorithms (cGAs), two parents might be selected by binary tournament, linear ranking, or fitness-proportionate selection applied to the (small) selection pool constituted by the cells belonging to the neighborhood of a given cell, including that cell itself. Recombination might be done on the selected individuals using a standard crossover technique, followed by mutation. However, multiparent techniques have also been proposed [106]. Finally, replacement of the cell might be done in several ways: the most common way is for one offspring to replace the cell if it is better, but a probabilistic choice or a policy of always replacing the cell is also common [70, 96].

Gorges-Schleuter gave extensions of the generic cEA to cover evolution strategies (ES) by using a local-pool form of the classical (λ, μ)-ES selection, in which, out of λ offspring, μ are chosen to form the next population [72]. Mutation and, possibly, recombination are the usual ES operators .

The phases of the cEA take place simultaneously for all cells in the lattice. On truly parallel hardware, this feature can be exploited to speed up the computation. On a single machine, the virtually parallel update is simulated by using an extra grid, which keeps the updated cells as they are produced. This grid will replace the old one at the end of one update phase, or generation. Remember, however, that here we are not discussing implementation issues, but only the models. Implementations are discussed in Appendix A.

In the study described here, we consider cEAs defined on a one-dimensional lattice of size n or a square lattice of size $m \times m$. Both the linear cEA and the two-dimensional case have periodic boundary conditions, i.e. the structures are a ring and a torus respectively; the latter is the topological structure depicted in Fig. 1.6.

Let us denote by S the (finite) set of states that a cell can take up or, equivalently, the set of different individuals that can occupy a cell at any given time: this is the set of points in the (discrete) search space of the problem. Let K_i be the set of neighbors of a given cell i, and let $|K_i| = K$ be its size. The local transition function $\phi(\cdot)$ can then be defined as

$$\phi : S^K \to S,$$

which maps the state $s_i \in S$ of a given cell i into another state from S, as a function of the states of the K cells in the neighborhood K_i. The implicit form of the stochastic transition function $\phi(\cdot)$ is

$$\phi(\cdot) = P(x_i(t+1) \mid x_j(t) \in K_i),$$

where P is the conditional probability that cell x_i will assume a certain value from the set S at the next time step $t+1$, given the current (at time t) values of the states of all the cells in the neighborhood. We are thus dealing with probabilistic automata, as was pointed out by Tomassini [146] and Whitley [157], and the set S should be seen as a set of values of a random variable. The probability P will be a function of the particular selection and variation methods used.

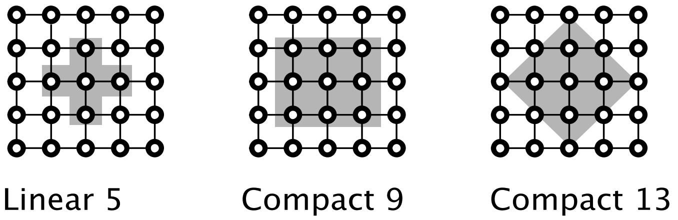


Fig. 4.2. Some neighborhoods used in cEA work. From left to right: Linear5, more commonly known as the von Neumann neighborhood in cellular automata work; Compact9, also called Moore; and Compact13, a radius-2 von Neumann neighborhood

The main neighborhoods that we consider in this chapter are the *radius-1* neighborhood in the linear case, which comprises the cell itself and its first right and left neighbors and, in the two-dimensional case, the von Neumann neighborhood, also called Linear 5 ($|N_i| = 5$), which is constituted by the central cell and the four first-neighbor cells in the directions north, east, south, and west (see Fig. 4.2). Other neighborhood structures are possible, and sometimes used, as depicted in the figure. For the sake of completeness, I should mention the fact that the neighborhood need not have a regular geometrical shape; sometimes, irregular neighborhoods have been used in which individuals are selected by performing short random walks around the central cell [33].

4.3 The Time Dimension: Asynchronous cEAs

Synchronous update, with its idealization of a global clock, is customary in cEAs and cellular automata, and most results have been obtained using this model. However, perfect synchronicity is only an abstraction: in physical or biological situations, the synchronicity assumption is untenable. In fact, in any spatially extended system, signals cannot travel faster than light. Hence, for given dimensions it is impossible for a signal emitted by a global clock to reach any two computing elements or agents at exactly the same time, which poses the problem of latching the signal for the units to work in synchronous mode. Indeed, in biological and sociological environments, agents normally act at different and possibly uncorrelated times, which seems to preclude a faithful globally synchronous simulation in most cases of interest (see, for example, [20] and [80]). Of course, this "unphysicality" is not a problem in artificial evolutionary algorithms, where we are free to use any solution that makes sense computationally, even when it is not defendable on biological grounds. But there are other reasons that make asynchronous cEAs potentially useful as problem solvers, as we shall see.

In the asynchronous case, cells are updated one at a time in some order. There are thus many ways for sequentially updating the cells of a cEA, including "mixed" ones in which whole blocks of cells are updated asynchronously with respect to each other, while cells belonging to the block are updated in parallel [137]. Here I consider four commonly used asynchronous update methods for cellular automata in which cells are are updated one by one [128]:

- In *fixed line sweep* (LS), the n cells are updated sequentially from left to right for rings, and line by line, starting from the upper left corner cell, for grids.
- In *fixed random sweep* (FRS), the next cell to be updated is chosen with uniform probability without replacement; this will produce a certain update sequence $(c_1^j, c_2^k, \ldots, c_n^m)$, where c_q^p means that cell number p is updated at time q and $(j, k, \ldots, m)$ is a permutation of the n cells. The same permutation is then used for all update cycles.
- The method of *new random sweep* (NRS) works like FRS, except that a new random cell permutation is used for each sweep through the array.
- In *uniform choice* (UC), the next cell to be updated is chosen at random with uniform probability and with replacement. This corresponds to a binomial distribution of the updating probability.

A *time step* is defined as the process of updating n times sequentially, which corresponds to updating *all* the n cells in the grid for LS, FRS, and NRS, and possibly fewer than n different cells in the uniform-choice method, since some cells might be updated more than once. Fixed line sweep is a rather degenerate updating policy, always imposing a one-by-one sequential scan of the array. In spite of this, it can be useful at times, and is also an interesting bounding asynchronous case to consider.

4.4 Models and Their Validation

There have been a number of investigations attempting to characterize the theoretical properties of cEAs. In [37] Davidor offered a schema propagation analysis of a two-dimensional cGA with local fitness-proportionate selection and a probabilistic replacement policy for the offspring. He showed that convergence within a neighborhood is rapid but, thanks to the slow diffusion of individuals through the grid from neighborhood to neighborhood, there is less disruption and the risk of premature convergence is lower.

Rudolph and Sprave [125] have shown how cGAs can be modeled by a probabilistic-automata network and have provided proofs of complete convergence to a global optimum based on Markov chain analysis for a model with a fitness threshold.

Several results have appeared on selection pressure in cEAs. Spiessens and Manderick [140] made some early guesses at the form of propagation of the individuals in a two-dimensional grid under local fitness-proportionate selection and suggested that the growth should follow a quadratic law. They also studied the time complexity of the cellular EA.

Sarma and De Jong performed empirical analyses of the dynamical behavior of cGAs [126, 127], focusing on the effect that the local selection method, and the size and shape of the neighborhood have on the global induced selection pressure. Recently, Giacobini et al. have successfully modeled the selection pressure curves on cEAs in one-dimensional rings and two-dimensional, torus-shaped grids [61, 62, 63, 65]. Here I shall follow this line of work, and the reader is referred to the original articles for more details.

In the next section I shall introduce some mathematical background that is needed in order to understand the models and, for the two-dimensional case, the approximations that are required to make the models workable. The models will then be described and validated experimentally. Finally, I shall offer some more general considerations on the effect of changing the grid shapes or scaling the neighborhood's size.

4.5 Mathematical Models

Let us consider the random variables $V_i(t) \in \{0, 1\}$ indicating the presence in cell i $(1 \leq i \leq n)$ of a copy of the best individual $(V_i(t) = 1)$ or of a worse one $(V_i(t) = 0)$ at time step t, where n is the the population size. The random variable

$$N(t) = \sum_{i=1}^{n} V_i(t) \tag{4.1}$$

denotes the number of copies of the best individual in the population at time step t. Initially $V_i(1) = 1$ for some individual i, and $V_j(1) = 0$ for all $j \neq i$.

Following Rudolph's definition [124], if the selection mechanism is nonextinctive, the expectation $E[T]$, where $T = \min\{t \geq 1 : N(t) = n\}$, is called the takeover time of the selection method. In the case of spatially structured populations the quantity $E_i[T]$, denoting the takeover time if cell i contains the best individual at time step 1, is termed the takeover time with initial cell i. Assuming a uniformly distributed initial position of the best individual over all cells, the takeover time is therefore given by

$$E[T] = \frac{1}{n}\sum_{i=1}^{n} E_i[T]. \tag{4.2}$$

In the following subsections, recurrences are given that describe the growth of the random variable $N(t)$ in cEAs with different regular lattice topologies for the synchronous and the four asynchronous update policies described in Sect. 4.2. We consider nonextinctive selection mechanisms that select the best individual in a given neighborhood with a probability in the interval $(0, 1]$.

4.5.1 Limitations of Logistic Modeling

It has been well known since the work of Verhulst[152] in the 19th century, that the assumption of logistic growth is a reasonable model for biological populations with bounded resources [110]. It is easy to see that this behavior also holds for the growth of the best individual in the artificial evolution of a finite panmictic population [67]. In fact, if we consider a population of size n, the number $N(t)$ of copies of the best individual in the population at time step t is given by the following recurrence:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + p_s N(t-1)(n - N(t-1)) \end{cases}$$

where p_s is the probability that the best individual is chosen. This recurrence can be easily transformed into one that describes a discrete logistic population growth in discrete time:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + p_s n N(t-1)\left(1 - (1/n)N(t-1)\right). \end{cases}$$

Such a recurrence can be approximated in analytic form by the standard continuous logistic equation[2]:

$$N(t) = \frac{n}{1 + (n/N(0) - 1)\, e^{-\alpha t}},$$

where the growth coefficient α depends on the probability p_s. This is the approach taken in [127] for synchronous cEAs in order to fit the measured growth curves as a function of a single structural parameter.

[2] Note that this is not true in a rigorous sense. The discrete logistic map can give rise to chaotic behavior for a range of the parameters [110]. This is ignored in the qualitative discussion above.

This can be useful as a first approximation, but, as suggested by Gorges-Schleuter and Spiessens and Manderick [72, 140], in the artificial evolution of locally interacting, spatially structured populations, the assumption of logistic growth does not hold anymore. Instead, in these locally interacting structures, although the curves have the familiar "S shape" denoting growth followed by saturation, they are not exponential but rather are polynomial, with a time dependence $\propto t^d$, where d is the lattice dimension.

In fact, in the case of a ring or a torus structure we have a linear or a sub-quadratic growth, respectively. We complete here Gorges-Schleuter's analysis, which holds for unrestricted growth, extending it to bounded synchronously updated spatial populations.

For a structured population, let us consider the limiting case, which represents an upper bound on the growth rate, in which the selection mechanism is deterministic (i.e. where $p_s = 1$), and a cell always chooses its best neighbor for updating. If we consider a population of size n with a ring structure, and consider a neighborhood radius of r (i.e. the neighborhood of a cell contains $2r + 1$ cells), the following recurrence describes the growth of the number of copies of the best individual:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 2r. \end{cases}$$

This recurrence can be described by the closed equation $N(t) = 1 + 2rt$, which clearly shows the linear character of the growth rate.

In the case of a population of size n on a toroidal grid of size $\sqrt{n} \times \sqrt{n}$ (assuming $\sqrt{n}$ odd) and a von Neumann generalized neighborhood structure of radius r (see Sect. 4.8), the growth of the number of copies of the best individual can be described by the following recurrence:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4\sum_{i=0}^{r-1}(rt - i) & , \quad 0 \le t \le (\sqrt{n} - 1)/2, \\ N(t) = N(t-1) + 4\sum_{i=0}^{r-1}(\sqrt{n} - rt - i) \ , & \quad t \ge (\sqrt{n} - 1)/2. \end{cases}$$

which reduces to the following:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4r^2 t - 2r(r+1) & , \quad 0 \le t \le (\sqrt{n} - 1)/2, \\ N(t) = N(t-1) - 4r^2 t + 4r\sqrt{n} - 2r(r+1) \ , & \quad t \ge (\sqrt{n} - 1)/2. \end{cases}$$

This growth is described by a convex quadratic equation followed by a concave one, as the two closed forms of the recurrence clearly show:

$$\begin{cases} N(t) = 2r^2t^2 + 2r(2r+1)t + 1 & , \quad 0 \leq t \leq (\sqrt{n}-1)/2, \\ N(t) = -2r^2t^2 + 2r(2\sqrt{n} - 3r - 1)t + 1 \ , & t \geq (\sqrt{n}-1)/2. \end{cases}$$

Figure 4.3 depicts graphically the growth described by the above equations for a population of 81 individuals on a 9×9 torus structure using a radius-1 von Neumann neighborhood.

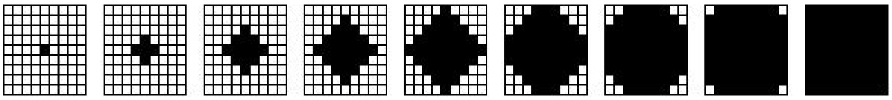


Fig. 4.3. Example of deterministic growth of $N(t)$ for a population of 81 individuals on a 9×9 torus structure with a von Neumann neighborhood

Thus, a more accurate fit should take into account the nonexponential growth followed by saturation (the crowding effect). We address such studies in the following sections for the cases of one- and two-dimensional regular lattice topologies with synchronous and asynchronous evolution.

4.5.2 The Ring Structure

In a linear cEA, the cells are arranged along a line. Depending on whether the last and first individuals communicate or not, we have a ring or a linear topology, respectively. Here we assume the first case (ring), which is more common. Each cell has the same number of neighbors on both sides, and this number depends on the *radius* r. We shall consider first the simplest case, $r = 1$, which means that there are three neighbors, including the central cell itself.

At each time step t, the expected number of copies $N(t)$ of the best individual is independent of its initial position. Therefore, the expected takeover time is $E[T] = E_i[T], \forall i$.

Synchronous Takeover Time

Since we are assuming neighborhoods of radius 1 and $N(0) = 1$, the set of cells containing a copy of the best individual will always be a connected region of the ring. Therefore, at each time step, only two more cells (the two adjacent to the connected region of the ring) will contain a copy of the best individual, with probability p. The growth of the quantity $N(t)$ can be described by the following recurrence:

$$\begin{cases} N(0) = 1, \\ E[N(t)] = \sum_{j=1}^{n} P[N(t-1) = j](j + 2p), \end{cases}$$

where $P[N(t-1)=j]$ is the probability that the random variable N takes the value j at time step $t-1$. Since $\sum_{j=1}^{n} P[N(t-1)=j]=1$, and the expected number $E[N(t-1)]$ of copies of the best individual at time step $t-1$ is by definition $\sum_{j=1}^{n} P[N(t-1)=j]j$, the above recurrence is equivalent to

$$\begin{cases} N(0)=1, \\ E[N(t)] = E[N(t-1)] + 2p. \end{cases}$$

The closed form of this recurrence is trivially $E[N(t)] = 2pt+1$, and therefore the expected takeover time $E[T]$ for a synchronous ring cEA with n cells is

$$E[T] = \frac{1}{2p}\,(n-1).$$

Rudolph [124] gave analytical results for a ring with synchronous update for a generic probability of selection p. Although obtained in a different way, the expression above and his equation give nearly the same results for large population sizes n. In fact, his equation, for large n, reduces to $n/2p-1/4$, while the equation above gives $n/2p-1/2p$. Since the first term quickly dominates over the second for large n, the two expressions can be considered equivalent.

Asynchronous Fixed-Line-Sweep Takeover Time

Let us consider the general case of an asynchronous fixed-line-sweep cEA, in which the connected region containing the copies of the best individual at time step t is $B(t)=\{l,\dots,k\}$, $1<l\le k<n$. At each time step the cell $l-1$ will contain a copy of the best individual with probability p, while the cells $k+j$ (with $j=1,\dots,n-k$) will contain a copy of the best individual with probability p^j. The recurrence describing the growth of the random variable $N(t)$ is therefore

$$\begin{cases} N(0)=1, \\ E[N(t)] = \displaystyle\sum_{j=1}^{n} P[N(t-1)=j]\left(j+p+\sum_{i=1}^{n-j} p^i\right). \end{cases}$$

Since $\sum_{i=1}^{n-j} p^i$ is a geometric progression, we can approximate this quantity for large n by the limit value $p/(1-p)$ of the sum. The recurrence is therefore equivalent to the following one:

$$\begin{cases} N(0)=1, \\ E[N(t)] = E[N(t-1)] + p + p/(1-p) = E[N(t-1)] + (2p-p^2)/(1-p). \end{cases}$$

Since the closed form of the recurrence above is

$$E[N(t)] = \frac{2p-p^2}{1-p}\,t+1,$$

we conclude that the takeover time for an asynchronous fixed-line-sweep cEA with a population of size n is

$$E[T] = \frac{1-p}{2p-p^2}(n-1).$$

Asynchronous Fixed- and New-Random-Sweep Takeover Times

The mean behaviors of the asynchronous fixed-random-sweep and new-random-sweep update policies over all the possible permutations of the sweeps are equivalent. I therefore give only one model, describing the growth of the random variable $N(t)$ for both policies.

Let us consider again the general case in which the connected region containing the copies of the best individual at time step t is $B(t) = \{l, \dots, k\}$ (with $1 < l \leq k < n$). The cells $l-1$ and $k+1$ have a probability p of containing a copy of the best individual at the next time step. For symmetry reasons, let us consider only the part of the ring at the right-hand side of the connected region. The cell $k+2$ has a probability $1/2$ of being contained in the set of cells after cell $k+1$ in the sweep, so it has a probability $(p/2)p$ of containing a copy of the best individual in the next time step. In general, a cell $k+j+1$ has a probability $1/2$ of coming after cell $k+j$ in the sweep, so it has a probability $(p/2)^j p$ to contain a copy of the best individual in the next time step. The recurrence describing the growth of the random variable $N(t)$ is therefore

$$\begin{cases} N(0) = 1, \\ E[N(t)] = \displaystyle\sum_{j=1}^{n} P[N(t-1) = j] \left(j + 2 \sum_{i=1}^{n-j} p \left(\frac{p}{2}\right)^{i-1} \right), \end{cases}$$

which can be transformed into the recurrence

$$\begin{cases} N(0) = 1, \\ E[N(t)] = \displaystyle\sum_{j=1}^{n} P[N(t-1) = j] \left(j + 4 \sum_{i=1}^{n-j} \left(\frac{p}{2}\right)^{i} \right). \end{cases}$$

Since $\sum_{i=1}^{n-j} (p/2)^i$ is a geometric progression, we can approximate this quantity for large n by the limit value $p/(2-p)$ of the sum. The recurrence is thus equivalent to the following one:

$$\begin{cases} N(0) = 1, \\ E[N(t)] = E[N(t-1)] + 4p/(2-p). \end{cases}$$

The closed form of the recurrence above is

$$E[N(k)] = \frac{4p}{2-p}\, k + 1,$$

and we conclude that the expected takeover time for a fixed- (or new-) random-sweep asynchronous cEA with a population of size n is

$$E[T] = \frac{2-p}{4p}\,(n-1).$$

Asynchronous Uniform-Choice Takeover Time

To model the takeover time for asynchronous uniform-choice cEAs, it is preferable to use cell update steps u instead of time steps in the recurrences. As in the case of the other update policies, the region containing the copies of the best individual at update step u is a connected part of the ring $B(u) = \{l, \dots, k\}$ (with $1 < l \leq k < n$). At each update step the two cells $l-1$ and $k+1$ have a probability $1/n$ of being selected, and each cell has a probability p, if selected, of containing a copy of the best individual after the selection and replacement phases. The recurrence describing the growth of the random variable $N(u)$ counting the number of copies of the best individual at update step u thus becomes

$$\begin{cases} N(0) = 1, \\ E[N(u)] = \displaystyle\sum_{j=1}^{n} P[N(u-1) = j]\left(j + 2\frac{1}{n}p\right), \end{cases}$$

which can be transformed into

$$\begin{cases} N(0) = 1, \\ E[N(u)] = E[N(u-1)] + 2p/n. \end{cases}$$

We can easily derive the closed form of the above recurrence

$$E[N(u)] = \frac{2}{n}\,pu + 1.$$

Since a time step is defined as n update steps, where n is the population size, the expected takeover time for a uniform choice asynchronous cEA is

$$E[T] = \frac{1}{2p}\,(n-1).$$

We notice that the expected takeover time for a uniform-choice asynchronous cEA is equal to the expected takeover time for a synchronous cEA.

It should be noted also that the present asynchronous uniform-choice update model is very similar to what goes under the name of *nonlinear voter model* in the probability literature [43].

4.5.3 The Torus Structure

We consider cEAs defined on a square lattice of finite size $\sqrt{n} \times \sqrt{n}$. The neighborhood is the von Neumann neighborhood, which is constituted by a central cell plus the four first-neighbor cells in the directions north, east, south, and west (see Figs. 4.1 and 4.2).

Because of the wrapping properties of the torus, at each time step t the expected number of copies $N(t)$ of the best individual is independent of its initial position. Therefore, the expected takeover time is $E[T] = E_i[T], \forall i$.

We have seen in Sect. 4.5.1 the limiting case of growth with deterministic selection (i.e. a mechanism that selects the best individual in the neighborhood with probability $p = 1$). In that case, the time variable t in the equations determines the half-diagonal of a square rotated by 45° (see Fig. 4.3). When a probabilistic selection method is modeled, the exact recurrences, corresponding to those derived for the ring topology in the previous subsection, become very complicated. In fact, as can be seen in Fig. 4.4, the phenomenon that has to be modeled implies different selection probabilities at different locations in the grid.

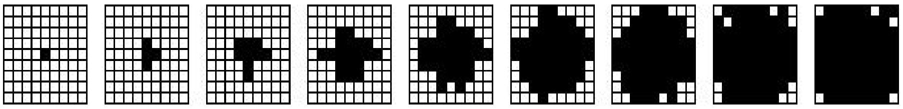


Fig. 4.4. Example of growth of $N(t)$ with probabilistic selection for a population of 81 individuals on a 9×9 torus structure

To keep the models simple and easily interpretable, the geometry of the propagation is approximated as the growth of a rotated square in the torus (see Fig. 4.5). Using this geometric growth, the side length s and the half-diagonal d of the rotated square can be approximated by

$$s = \sqrt{N(t)}, \quad d = \frac{\sqrt{N(t)}}{\sqrt{2}}.$$

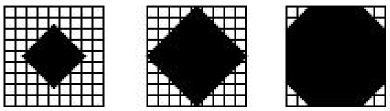


Fig. 4.5. Geometric approximation of growth with probabilistic selection in a torus-structured population: a rotated square grows as a function of time; there is unrestricted growth until the square reaches the edges of the grid, and then the population saturates

With these quantities, we shall now focus on synchronous and asynchronous takeover times, using the relevant probabilities in each case.

Synchronous Takeover Time

Let us consider the growth of such a region with a selection mechanism that has probabilities p_1, p_2, p_3, p_4, and p_5 of selecting the best individual when there are respectively 1, 2, 3, 4 and 5 copies of it in the neighborhood. Assuming that the region containing the copies of the best individual expands such that it maintains the shape of a square rotated by 45°, we can model the growth of $N(t)$ with the following recurrence

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4p_2\sqrt{N(t-1)}/\sqrt{2}\ , & N(t) \leq n/2, \\ N(t) = N(t-1) + 4p_2\sqrt{n - N(t-1)}\ , & N(t) > n/2. \end{cases}$$

It is extremely difficult to find a closed analytic form of this recurrence, as is also the case for the asynchronous models considered next. Therefore, only the explicit recurrences will be given in each case.

Asynchronous Fixed-Line-Sweep Takeover Time

This update method, which is meaningful in a ring topology, can be criticized in the case of a toroidal topology. In fact, there is no biological parallel for this update mechanism. A precise model for such an update would be very complicated, since it is difficult to approximate the shape of the region containing the copies of the best individual. In order to keep the model simple and understandable, the shape of the growing region is approximated here by a square stretched in the southeast direction, growing with probability p_1 on the northeast side, p_2 on the southeast side, and p_1 in the southerly direction.

Let us suppose that in any line the cells containing a copy of the best individual at time step t have indices l to k. In the next time step, the cell $l-1$ will contain a copy of the best individual with probability p, while the cells $k+j$ (with $j = 1, \ldots, n-s$) will contain a copy of the best individual with probability p^j. The number of copies of the best individual in the line in the next time step is

$$p + \sum_{i=1}^{\sqrt{n}-j} p^i .$$

For large n, we can approximate this quantity by the limit $(2p - p^2)/(1-p)$. Therefore, we can model the growth of $N(t)$ with the following recurrence

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + \left((2p_2 - p_2^2)/(1-p_2) + 2(2p_1 - p_1^2)/(1-p_1)\right)\sqrt{N(t-1)}, \\ \qquad N(t) \le n/2, \\ N(t) = N(t-1) + \left((2p_2 - p_2^2)/(1-p_2) + 2(2p_1 - p_1^2)/(1-p_1)\right) \\ \qquad \sqrt{n - N(t-1)}, \quad N(t) > n/2. \end{cases}$$

Asynchronous Fixed- and New-Random-Sweep Takeover Time

The behaviors of fixed random sweep and new random sweep averaged over all the possible permutations of the individuals on the grid are equivalent in the toroidal case also. Thus, only one model is needed to describe the growth of the random variable $N(t)$ for both policies.

In any one time step, the probability of one individual on the border of the region being taken over by the best is p_2, while an individual at distance 2 from the region can be replaced by the best with probability (details can be found in [65])

$$p_2 p_1 + \frac{1}{4}(p_2 - 2p_1)p_2^2.$$

It is sufficient to model the growth up to distance 2 because, as can been seen in Fig. 4.6, the probability at distances ≥ 3 becomes negligible.

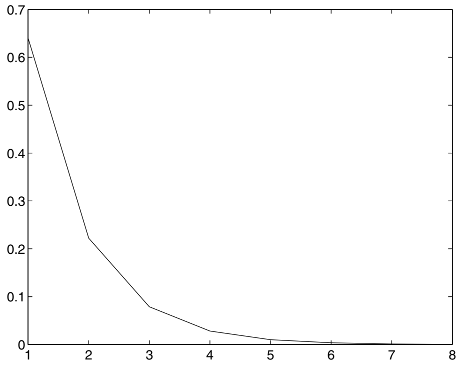


Fig. 4.6. Probability of an individual being replaced by a copy of the best individual (y axis) as a function of the distance (x axis) from the region formed by copies of the best, for asynchronous fixed (and new) random sweep. Note that the curve is traced continuously for clarity but the probability is calculated only at discrete points

Thus, we can model the growth of $N(t)$ with the following recurrence:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4\left(p_2p_1 + \frac{1}{4}(p_2 - 2p_1)p_2^2\right)(\sqrt{N(t-1)} - 1) + 4p_1, \\ \qquad N(t) \leq n/2, \\ N(t) = N(t-1) + 4\left(p_2p_1 + \frac{1}{4}(p_2 - 2p_1)p_2^2\right)(\sqrt{n - N(t-1)} - 1) + 8p_3, \\ \qquad N(t) > n/2. \end{cases}$$

Asynchronous Uniform-Choice Takeover Time

The ways in which an individual can be replaced in a time step in this case are the same as for fixed and new random sweep (see above). In the present case, the average probability of an individual coming before another in a time step is $1/n$; therefore, an individual at distance 2 from the region is replaced with probability

$$\frac{1}{n}p_2p_1 + \frac{1}{n^2}(p_2 - 2p_1)p_2^2.$$

The probability is already very small at distance 2 (see Fig. 4.7). Thus, only individuals at distance 1 from the region are considered.

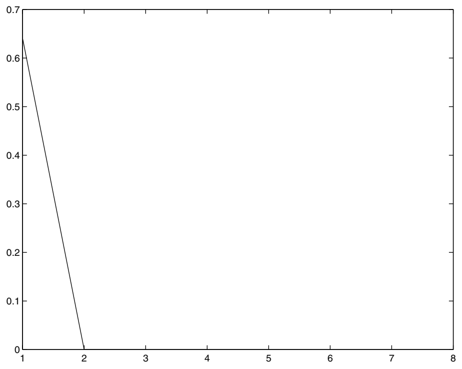


Fig. 4.7. Probability of an individual being replaced by a copy of the best individual (y axis) as a function of the distance (x axis) from the region formed by copies of the best for uniform choice. Note that the curve is traced continuously for clarity but the probability is calculated only at discrete points

In terms of time steps, the growth of $N(t)$ can be modeled with the following recurrence:

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4p_2\sqrt{N(t-1)} & , \quad N(t) \leq n/2, \\ N(t) = N(t-1) + 4p_2(\sqrt{n - N(t-1)} - 1) + 8p_3 & , \quad N(t) > n/2 \end{cases}$$

4.6 Experimental Validation

In this section, a set of validation tests to evaluate the accuracy of the mathematical models in the previous sections are presented. Since cEAs are good candidates for using selection methods that are easily extensible to small local pools, we have used binary tournament and linear ranking in the experiments. Fitness-proportionate selection could also be used, but it suffers from stochastic errors in small populations (e.g. at a neighborhood level), and it is more difficult to model theoretically, since it requires knowledge of the fitness distribution function.

The binary tournament selection mechanism is the same as the one described by Rudolph [124]: two individuals are randomly chosen with replacement in the neighborhood of a given cell, and the one with the better fitness is selected for the replacement phase.

In linear ranking selection the individuals in the neighborhood of a given cell are ranked according to their fitness: each individual then has a probability $2(s-i)/(s(s-1))$ of being selected for the replacement phase, where s is the number of cells in the neighborhood and i is its rank in the neighborhood.

4.6.1 Ring Structure

For the study described here, the cEA structure has a ring topology of size 1024 with neighborhood of radius 1. Only the selection operator is active: for each cell, it selects one individual in the neighborhood of the cell (the cell and its two adjacent right and left neighbors). The selected individual replaces the old individual only if it has a better fitness.

Binary Tournament Selection

Figure 4.8 shows the experimental growth curves of the best individual for the synchronous and four asynchronous update methods. We may notice that the mean curves for the two asynchronous methods fixed and new random sweep show a very similar behavior. The graph also shows that the asynchronous update methods give an emergent selection pressure greater than or equal to that in the synchronous case, increasing from the case of uniform choice to that of line sweep, with fixed and new random sweep in between.

The numerical values of the mean takeover times for the five update methods, along with their standard deviations, are shown in Table 4.1, where it can be seen that the fixed-random-sweep and new-random-sweep methods give results that are statistically indistinguishable, and can therefore be described by a single model, as we assumed in Sect. 4.5.2. The same can be said for the synchronous and uniform-choice methods, as our models predicted.

Since the neighborhood has radius 1, at most one individual with the best fitness will be present in the neighborhood of any cell under consideration, except for the last update, when there are two of them. It turns out that

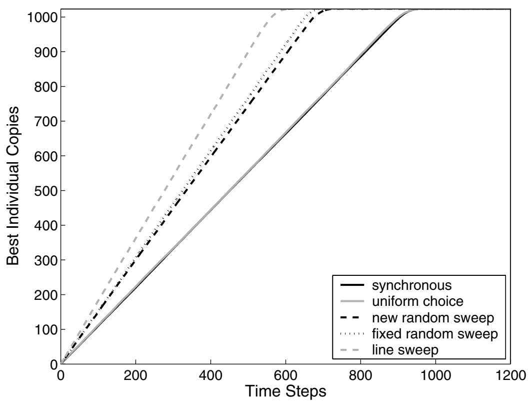


Fig. 4.8. Takeover times with binary tournament selection: mean values over 100 runs. The vertical axis represents the number of copies $N(t)$ of the best individual in each population as a function of the time step t

	Synchro	LS	FRS	NRS	UC
Mean takeover time	925.03	569.82	666.18	689.29	920.04
Standard deviation	20.36	24.85	17.38	20.27	26.68

Table 4.1. Mean actual takeover time and standard deviation for tournament selection and the five update methods. Mean values over 100 independent runs

the probability for an individual that has a copy of the best individual in its neighborhood being selected is $p = 5/9$. Using this probability in the models described in Sect. 4.1, theoretical growth curves can be calculated. Figure 4.9 shows the predicted and experimental curves for the five update methods, and the mean square error between them.

Looking at the curves, it is clear that the models faithfully predict the observed takeover times. Moreover, the equivalence between new random sweep and fixed random sweep, as well as that between synchronous and uniform choice, is fully confirmed.

Linear Ranking Selection

Figure 4.10 shows the experimental growth curves of the best individual for the synchronous and four asynchronous update methods. We can observe in the linear ranking case the same behavior as that which previously emerged in the binary tournament case: the mean curves for the cases of synchronous and asynchronous uniform choice are superposed, and the mean curves for the two asynchronous methods for fixed and new random sweep show very similar behavior. The graph shows that the asynchronous update methods give a greater emergent selection pressure than do the synchronous methods,

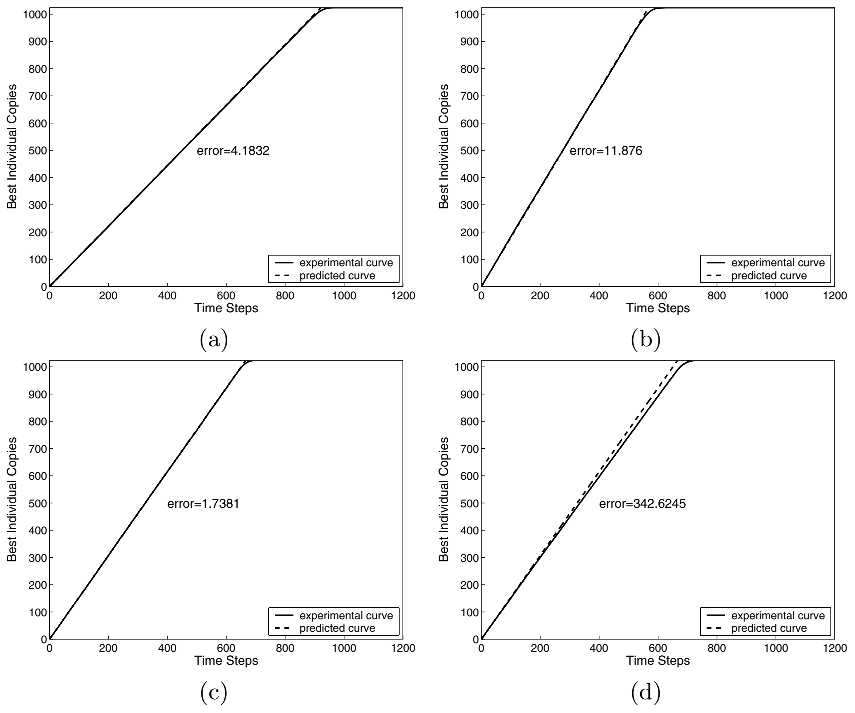


Fig. 4.9. Comparison of the experimental takeover time curves (full lines) with the model (dashed) in the case of binary tournament selection for four update methods: synchronous (a), asynchronous line sweep (b), asynchronous fixed random sweep (c), and asynchronous new random sweep (d). Asynchronous uniform choice gives the same curve as does synchronous update, and therefore the corresponding curve has been omitted. In each graph, the mean square error between the predicted and experimental curves is shown

increasing from the case of uniform choice to that of line sweep, with fixed and new random sweep in between.

The numerical values of the mean takeover times for the five update methods, along with their standard deviations, are shown in Table 4.2. Again, the results show that the two random-sweep methods are statistically equivalent, which is also the case for the synchronous and uniform-choice methods.

	Synchro	LS	FRS	NRS	UC
Mean takeover time	768.04	387.09	519.92	541.14	766.5
Standard deviation	17.62	19.21	14.26	14.48	25.44

Table 4.2. Mean takeover time and standard deviation for linear ranking selection and the five update methods. Mean values over 100 independent runs

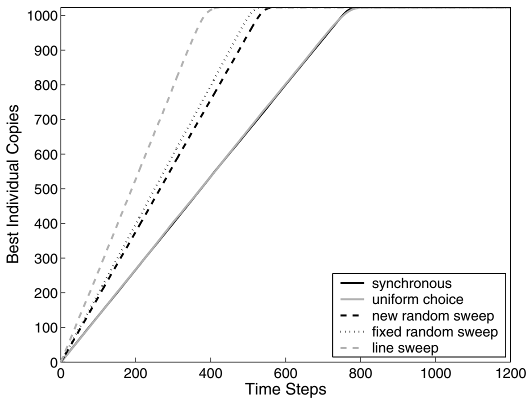


Fig. 4.10. Takeover times with linear ranking selection: mean values over 100 runs. The vertical axis represents the number of copies $N(t)$ of the best individual in each population as a function of the time step t

With this linear ranking selection method, a cell that has a copy of the best individual in its neighborhood has a probability $p = 2/3$ of selecting it. Using this value in the models described in Sect. 4.1, theoretical growth curves can be calculated. Figure 4.11 shows the predicted and experimental curves for the five update methods, and the mean square error between them. As can be seen, the agreement between theory and experiment is excellent.

4.6.2 Torus Structure

This section describes the validation of the models for the torus shape. The cEA structure used has a torus topology of size 32×32 with a von Neumann neighborhood. Only the selection operator is active: for each cell, it selects one individual in the neighborhood of the cell, and the selected individual replaces the old individual only if it has a better fitness. Results are presented for the two selection methods of binary tournament and linear ranking.

Binary Tournament Selection

Figure 4.12 shows the growth curves of the best individual for the panmictic, synchronous, and three asynchronous update methods. The mean curves for the two asynchronous methods, i. e. fixed and new random sweep, show a very similar behavior, and thus only the results for new random sweep are plotted. The graph shows that the asynchronous update methods give an emergent selection pressure greater than in the synchronous case, increasing from the case of uniform choice to that of line sweep, with fixed and new random sweep in between (similarly to our findings for the ring topology).

The numerical values of the mean takeover times for the five update methods, together with their standard deviations, are shown in Table 4.3, where

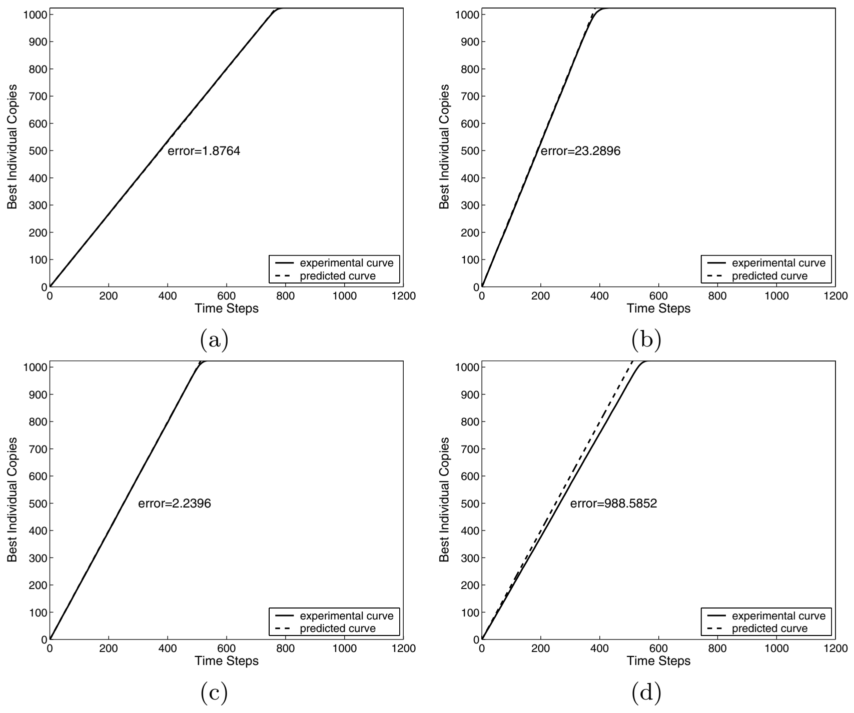


Fig. 4.11. Comparison of experimental takeover time curves (full lines) with the model (dashed) in the case of linear ranking selection for four update methods: synchronous (a), asynchronous line sweep (b), asynchronous fixed random sweep (c), and asynchronous new random sweep (d). Asynchronous uniform choice gives the same curve as does synchronous update, and therefore the corresponding curve has been omitted. In each graph, the mean square error between the predicted and experimental curves is shown

it can be seen that the fixed-random-sweep, and new-random-sweep methods give results that are statistically indistinguishable. However, this time the differences between the uniform-choice and synchronous update are meaningful in the case of torus.

	Synchro	LS	FRS	NRS	UC
Mean takeover time	44.06	21.8	27.21	28.26	35.73
Standard deviation	1.6746	1.7581	1.5654	1.8996	2.4489

Table 4.3. Mean takeover time and standard deviation for the binary tournament selection and the five update methods. Mean values over 100 independent runs

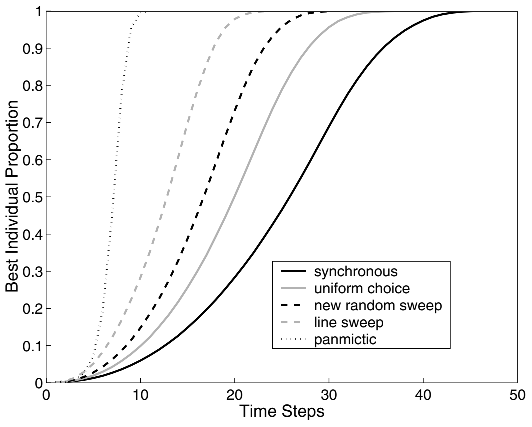


Fig. 4.12. Takeover times with binary tournament selection. Mean values over 100 runs. The vertical axis represents the fraction of the best individual in each population as a function of the time step t

Since a von Neumann neighborhood is used, the probabilities p_1, p_2 and p_3 of selecting the best individual when there are 1, 2, and 3 copies of it in the neighborhood are 9/25, 16/25, and 21/25, respectively. Using these probabilities theoretical growth curves can be calculated from the growth equations. Figure 4.13 shows the predicted and experimental curves for the five update methods. It can be observed that the agreement between theory and experiment is very good, in spite of the approximations made in the models.

Linear Ranking Selection

Figure 4.14 shows the growth curves of the best individual for the panmictic, synchronous, and three asynchronous update methods for linear ranking selection. We can observe the same phenomenon as that which emerged in the binary tournament case: the average curves for the two asynchronous update methods of fixed and new random sweep show very similar behavior. Thus, only the results for new random sweep are plotted. The graph shows that the asynchronous update methods give an emergent selection pressure greater than that for the synchronous method, increasing from the case of uniform choice to that of line sweep, with fixed random sweep in between. The numerical values of the mean takeover times for the five update methods, together with their standard deviations, are shown in Table 4.4. Again, the results show that the two random-sweep methods are statistically equivalent, while the uniform-choice and synchronous methods are not.

For a von Neumann neighborhood, the probabilities p_1, p_2, and p_3 of selecting the best individual when there are 1, 2, and 3 copies of it in the neighborhood are 2/5, 7/10 and 9/10, respectively. Theoretical growth curves

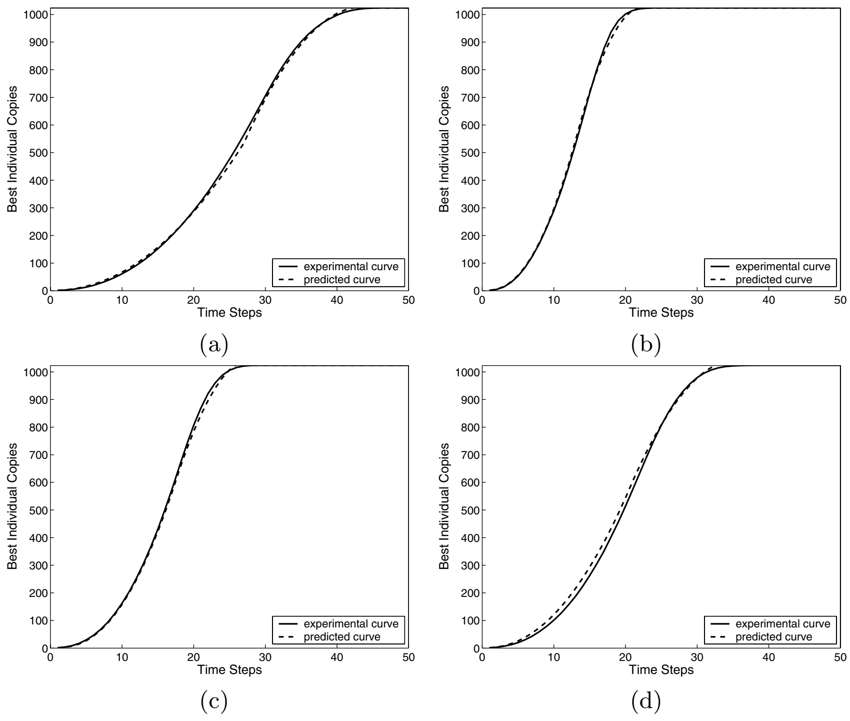


Fig. 4.13. Comparison of experimental takeover time curves (full line) with the model (dashed) in the case of binary tournament selection for four update methods: synchronous (a), asynchronous line sweep (b), asynchronous fixed random sweep (c), and uniform choice (d)

	Synchro	LS	FRS	NRS	UC
Mean takeover time	40.68	18.2	23.96	24.89	32.16
Standard deviation	1.2703	1.633	1.4766	1.4626	2.3856

Table 4.4. Mean takeover time and standard deviation for linear ranking selection and the five update methods. Mean values over 100 independent runs

can be calculated using these probability values in the models. Figure 4.15 shows the predicted and experimental curves for the five update methods. The agreement between theory and experiment can be considered very good.

4.7 Rectangular Toroidal Structures

It has been shown in the literature [4, 6, 41] that varying the ratio of the grid axes in a two-dimensional cEA is another simple way to control the

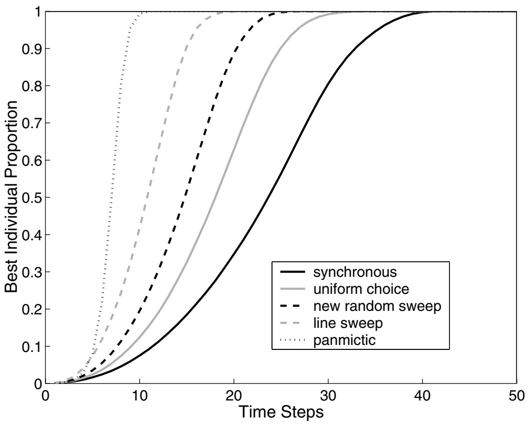


Fig. 4.14. Takeover times with linear ranking. Mean values over 100 runs. The vertical axis represents the fraction of the best individual in each population as a function of the time step t

global selection pressure. In this section, we address the prediction of takeover regimes for cEAs whose population shape is toroidal but not square. Since different kinds of rectangular shapes could be used in a toroidal cEA, I present the model's behavior when this variation is taken into account.

Let us suppose a rectangular toroidal structure of size $a \times b$, with $a \geq b$; the same geometrical approximation used in the case of a square toroidal structure (see Fig. 4.5) can be applied in this case. This time, the models will describe the growth of a rotated square until its area is equal to $b^2/2$, followed by a composition of b linear growths until the area of the region is $ab - b^2/4$, and then a final quadratic saturation (see Fig. 4.16).

The recurrences modeling the synchronous and the three asynchronous evolutions will therefore be made up of the initial condition ($N(0) = 1$), followed by the equation describing the unrestricted growth of the square (until $N(t) = b^2/2$), then the composition of b linear growths (until $N(t) = ab - b^2/4$), and finally the saturation equation. Since the expressions are rather cumbersome, only the final recurrence for the synchronous case is given here. The interested reader can find the detailed derivations, including the asynchronous cases, in [65].

For synchronous evolution the recurrence is

$$\begin{cases} N(0) = 1, \\ N(t) = N(t-1) + 4p_2(\sqrt{N(t-1)}/\sqrt{2}), \\ N(t) = N(t-1) + 2(b-1)p_2 + p_1, \\ N(t) = N(t-1) + 4p_2\sqrt{n - N(t-1)}. \end{cases}$$

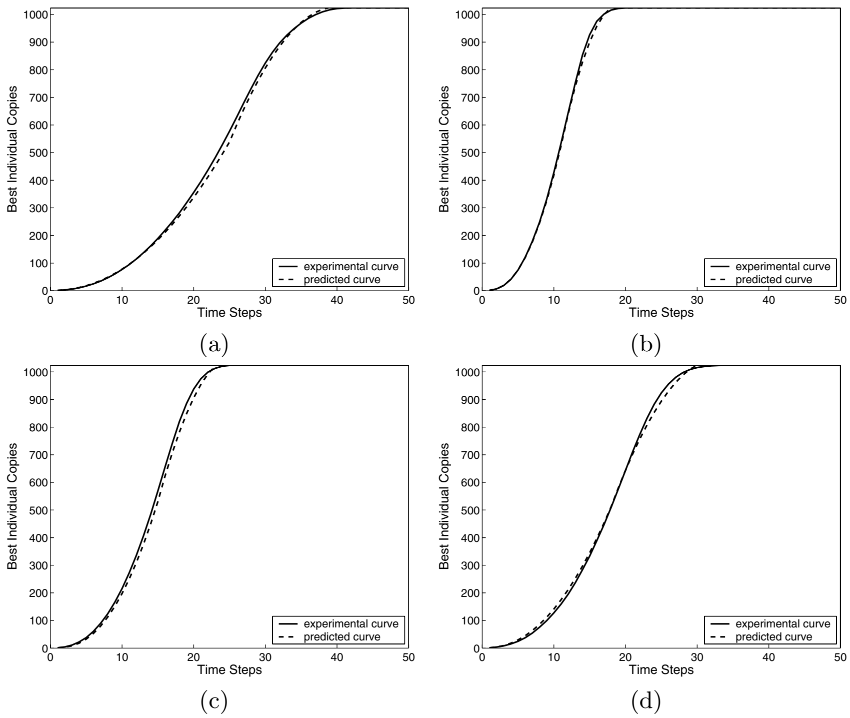


Fig. 4.15. Comparison of experimental takeover time curves (full lines) with the model (dashed) in the case of linear ranking selection for four update methods: synchronous (a), asynchronous line sweep (b), asynchronous fixed random sweep (c), and uniform choice (d)

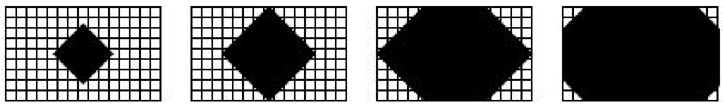


Fig. 4.16. Geometric approximation of growth with probabilistic selection in a rectangular toroidal structured population

4.7.1 Validation of the Rectangular Toroidal Models

The cEA structures used for this purpose had rectangular torus topologies of sizes 64×16 and 128×8 with a radius-1 von Neumann neighborhood. The cEA was run for the two topologies as in the previous experiments, using binary tournament and the linear ranking selection. The results (summarized in Table 4.5 for binary tournament and in Table 4.6 for linear ranking selection) show similar behavior to the ring and torus topologies. In fact, the longest takeover time always corresponds to a synchronous update, and the takeover times for the four asynchronous evolutions are ranked with fixed line sweep being the fastest, uniform choice being the slowest, and fixed and new random sweeps, which are statistically indistinguishable, in between.

	Synchro	LS	FRS	NRS	UC
64×16	62.15 (2.4)	29.99 (2.3)	38.1 (1.9)	39.37 (2.0)	48.96 (2.9)
128×8	117.07 (3.7)	55.37 (4.2)	70.57 (3.3)	73.26 (3.5)	89.48 (4.3)

Table 4.5. Mean takeover time, with standard deviation in parentheses, for binary tournament selection and the five update methods on 64×16 and 128×8 rectangular toroidal topologies. Mean values over 100 independent runs

	Synchro	LS	FRS	NRS	UC
64×16	57.42 (2.1)	24.6 (2.2)	33.89 (2.1)	35.3 (1.9)	45.05 (2.9)
128×8	108.32 (3.1)	45.98 (3.5)	62.69 (2.6)	64.76 (3.0)	80.78 (4.6)

Table 4.6. Mean takeover time, with standard deviation in parentheses, for linear ranking selection and the five update methods on 64×16 and 128×8 rectangular toroidal topologies. Mean values over 100 independent runs

As expected, the selection pressure induced using different grid sizes reduces as the grid becomes thinner [6, 41]. Figure 4.17 shows the different growth curves that result when the values of the axes of the grid are changed from those for a square to those for a thin rectangle, when the binary tournament and the linear ranking selection schemes are used.

As it can be seen in Figs. 4.18 and 4.19, the models successfully predict the experimental curves. When linear ranking selection is used, the accuracy of the models is comparable, but the curves are not shown for brevity.

4.8 Varying the Radius

Sarma and De Jong have shown convincingly that the neighborhood's size and shape have an important influence on the induced global selection pressure in grid-structured populations [126, 127]. To complete the study, in this section the basic value of 1 used for the radius until now is changed by using a generalized von Neumann neighborhood of radius 2 in both one- and two-dimensional regular lattices. Such a neighborhood is defined as containing all the individuals at distances smaller than or equal to the radius, where the Manhattan distance is used on the two-dimensional grid.

The equation for a ring with radius 2 is as follows:

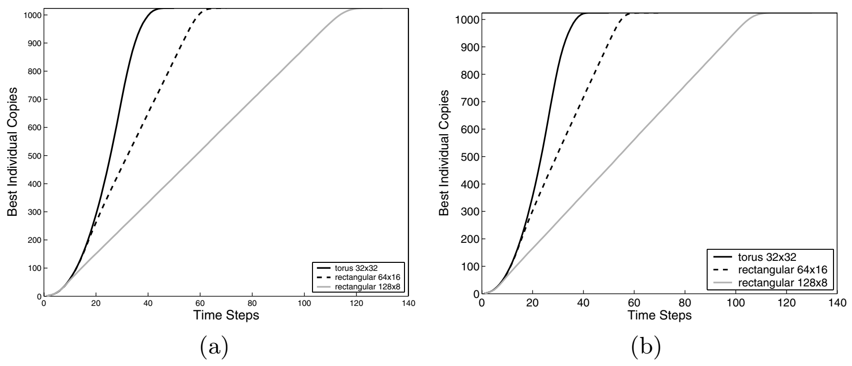


Fig. 4.17. Growth curves for synchronous evolution on toroidal structures with different ratios of axes using (a) binary tournament selection, and (b) linear ranking selection. Mean values over 100 independent runs

$$\begin{cases} N(0) = 1, \\ N(1) = N(0) + 4p_1, \\ N(t) = N(t-1) + 2p_2 + 2p_1, \end{cases}$$

where p_i is the probability that a copy of the best individual is selected when i copies of it are present in the neighborhood.

Using the same geometrical approximation described in Sect. 4.5.3, it is possible to obtain the model equations for a torus with a radius-2 generalized von Neumann neighborhood. However, since the expressions are rather involved, they are omitted here. Details can be found in [65].

The models have been tested using a binary tournament selection mechanism: comparisons between the predicted and actual curves in the case of synchronous update are shown in Fig. 4.20. The models are still accurate, and could be extended to larger-radius generalized von Neumann neighborhoods.

Although it is not apparent from the explicit recurrences, one can suppose that when the radius tends to $n/2$ in the ring case, and when it tends to $\sqrt{n}$ in the grid case, then the curves tend to the panmictic limit. This can be clearly seen in Fig. 4.21 a and Fig. 4.21 b for the ring and torus cases respectively, where experimental curves for several radii are plotted.

4.9 Summary

In this chapter we have seen mathematical models that can accurately predict the growth curves and the takeover time regime and values for a broad range of cellular evolutionary algorithms on regular lattices. Both rings and toroidal topologies have been studied. Two selection methods that are often used with

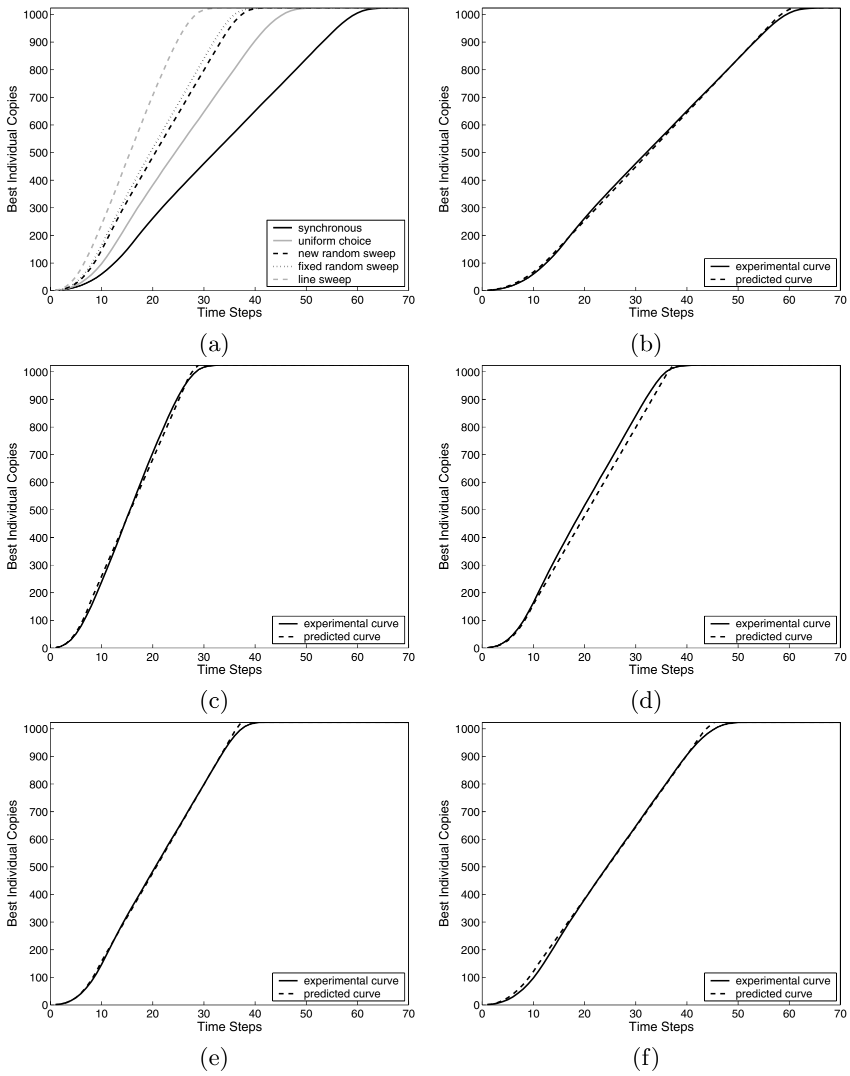


Fig. 4.18. Experimental curves for a rectangular 64 × 16 topology with binary tournament (a). Comparison of experimental takeover time curves (full lines) with the model (dashed) in the case of linear ranking selection for five update methods: synchronous (b), asynchronous line sweep (c), asynchronous fixed random sweep (d), asynchronous new random sweep (e), and uniform choice (f)

small selection pools have been used for the experimental validation of the models: tournament and linear ranking. Other selection methods, such as (μ, λ) or $(\mu + \lambda)$ used in evolution strategies could also be studied, since the

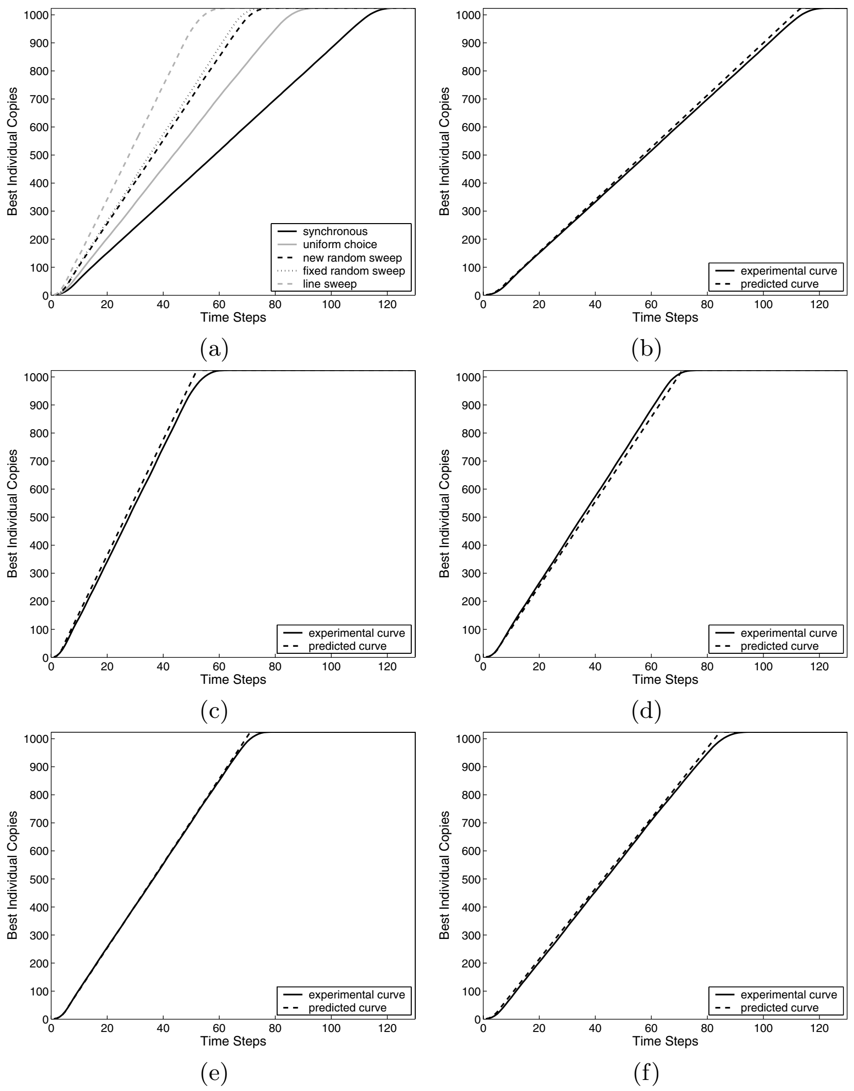


Fig. 4.19. Experimental curves for a rectangular 128×8 topology with binary tournament (a). Comparison of the experimental takeover time curves (full lines) with the model (dashed) in the case of linear ranking selection for five update methods: synchronous (b), asynchronous line sweep (c), asynchronous fixed random sweep (d), asynchronous new random sweep (e), and uniform choice (f)

models need only the actual probability of selection to be inserted into the equations.

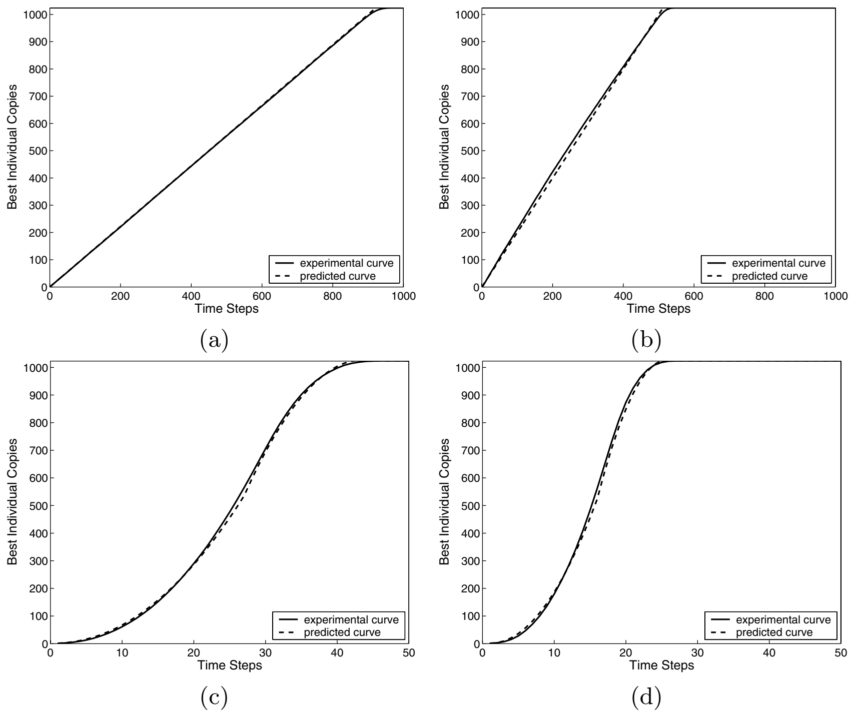


Fig. 4.20. Comparison of experimental takeover time curves (full lines) with the model (dashed) in the case of binary tournament selection for synchronous update in the case of a ring with radius-1 (a) and radius-2 (b) neighborhoods, and for a torus with radius-1 (c) and with radius-2 (d) generalized von Neumann neighborhoods

The main observation, already noted before by several researchers, is that the selection intensity in the population is lower in lattices than in panmictic populations. We have also seen, and this was previously unknown since only the customary synchronous update method was used, that different asynchronous policies give rise to significantly different global emergent selection pressures, and can thus be used to control the explorative or exploitative character of the algorithm to some extent, without resorting to ad hoc tricks in the selection methods.

Another way that can be used to control the selection intensity in cEAs is through the size and shape of the neighborhood or, in the two-dimensional case, by statically or dynamically adapting the ratio of the grid axes, with "flatter" rectangular shapes giving lower global pressures.

In conclusion, cellular evolutionary algorithms on regular lattices are a simple and straightforward way of structuring the population that allows the selection pressure to be controlled to an extent that makes tuning of the algorithm to the problem easier. Thus, they offer a high degree of flexibility in

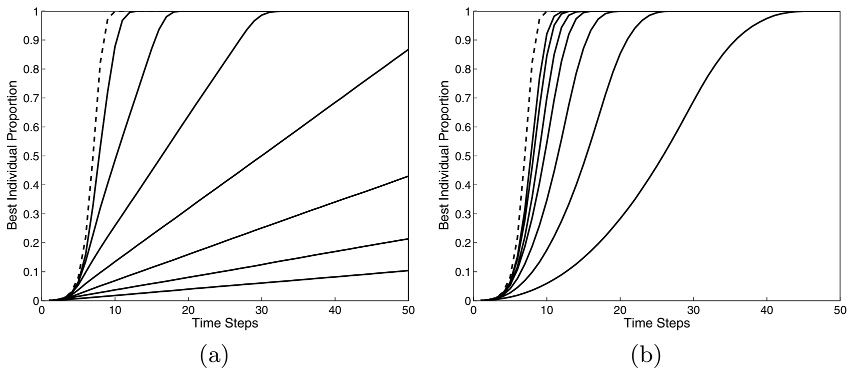


Fig. 4.21. (a) Growth curves for rings with neighborhoods of increasing radius. From right to left the radii are 2, 4, 8, 16, 32, 64, and 128. (b) Growth curves for tori with neighborhoods of radius 1, 2, 3, 4, 5, 6, and 7, increasing from right to left. The dashed curves in (a) and (b) pertain to the case of a panmictic population

problem-solving. In the next chapter, we shall see how these considerations can be put into practice, by empirically studying a number of test problems with cEAs.

5 Lattice Cellular Models: Empirical Properties

The previous chapter has laid the foundations for cellular evolutionary algorithms for populations structured as regular lattices. In the present chapter I would like to show how these lattice cEAs behave in practice on a number of benchmark problems. Ideally, we would like the theoretical trends to be confirmed by the computer experiments. This has been the case for the global emergent selection intensity. However, now the picture becomes more complex and more realistic, since we are studying full-fledged cEAs with selection, as well as variation operators.

Several articles on the application of cEAs to optimization problems have appeared, some of which are listed in the references (e.g. [16, 69, 71]). To illustrate the main ideas, I shall use here some material from two previous studies, one on cGAs [41], and the other on cellular genetic programming (cGP) [57]. This chapter is intended to be self-contained but the original references have more details, should the reader wish to study the issues more deeply.

Following the analysis in the previous chapter, the main points that I would like to illustrate here are the influence of the update mode and the grid shape on the character of the search. Concerning synchronous and asynchronous cell update policies, there is nothing new: these policies will be identical to those introduced in Chap. 4. For the influence of grid shape influence, I shall use the concept of a "rectangular grid" that was analyzed earlier, and the ratio of the grid axes.

5.1 cGA Case Study

The algorithmic schema employed in the study is similar to the one that appears in Sect. 4.2; it is just specialized to GAs:

```
for each cell $i$ in the grid do in parallel
    generate a random individual $i$
end parallel for
while not termination condition do
    for each cell $i$ in the grid do in parallel
        Evaluate individual $i$
        Select individuals in the neighborhood $N(i)$ by binary tournament
        Recombine individuals with double-point crossover
        Mutate offspring
        Evaluate offspring(s)
        Assign the best offspring to cell $i$
    end parallel for
end while
```

This is the classical synchronous cEA. The asynchronous variants are similar, except that the updating of the cells does not take place virtually simultaneously but rather in some specified sequential order, as explained in the previous chapter.

5.2 Varying the Lattice Shape

After explaining the basic algorithm in the previous section, we now proceed to characterize the population grid itself. In the previous chapter, some models were derived to take account of nonsquare grids. These are acceptable but, since they are expressed as elementary recurrences, they do not contain an explicit dependence on any parameter that gives the amount of "flatness" of the two-dimensional grid. For this reason, here I prefer to use the definition of "radius" given by Alba and Troya [6], which is refined from the seminal definition given in [126], to take account of nonsquare grids. The grid is considered to have a radius equal to the dispersion of n^* points on a circle centered in $(\overline{x}, \overline{y})$ (Eq. 5.1):

$$rad = \sqrt{\frac{\sum (x_i - \overline{x})^2 + \sum (y_i - \overline{y})^2}{n^*}}, \tag{5.1}$$
$$\overline{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*}, \; \overline{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*}$$

This definition always assigns different numerical values to different grids. Although it is called a "radius", *rad* measures the dispersion of n^* patterns. Other measures for symmetrical topologies would allocate the same numerical value to different topologies (which is undesirable). The definition (5.1) not only characterizes the grid shape but can also provide a value of the radius for the neighborhood. As proposed in [126], the grid-to-neighborhood relationship can be quantified by the ratio of their radii:

$$ratio_{cEA} = \frac{rad_{Neighborhood}}{rad_{Topology}}. \tag{5.2}$$

When we are solving a given problem with a constant number of individuals ($n = n^*$, to make fair comparisons), the radius of the topology will increase as the grid becomes thinner (Fig. 5.1b). Since the neighborhood is kept constant in size and shape throughout the study described here (we always use Linear 5, also called the von Neumann neighborhood, Fig. 5.1a), the ratio will become smaller as the grid becomes thinner.

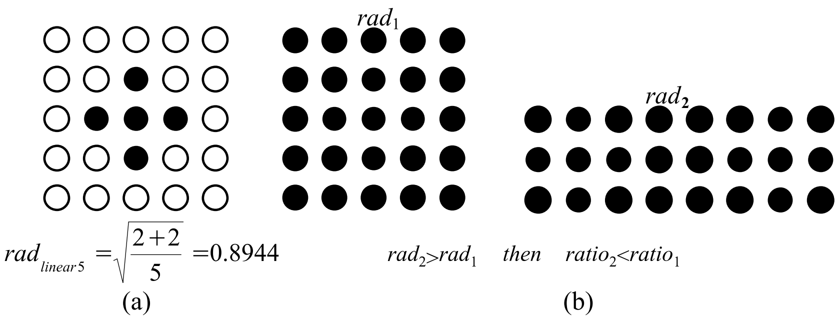

rad_1
rad_2
$rad_{linear5} = \sqrt{\frac{2+2}{5}} = 0.8944$
$rad_2 > rad_1$ then $ratio_2 < ratio_1$
(a)
(b)

Fig. 5.1. (a) Radius of von Neumann neighborhood. (b) $5 \times 5 = 25$ and $3 \times 8 = 24$ grids. Approximately equal number of individuals with two different ratios

After we have presented this characterization of the radius and topology by means of a ratio, the main question still remains: what is the importance of such a ratio measure? The answer, of course, is that, as we saw in the previous chapter, reducing the ratio means reducing the global selection intensity on the population, thus promoting *exploration*. This is expected to allow a higher diversity in the genotypic pool, which could help improve results in difficult problems such as multimodal or epistatic problems. On the other hand, the search performed inside each neighborhood guides the *exploitation* of the algorithm. Thus, we want to investigate how the ratio affects the search efficiency over a variety of domains.

5.3 Selection Pressure, Grid Shape and Time

Here I shall recall a few concepts from the previous chapter for ease of reference. There we saw that synchronous and a few asynchronous cell update policies give rise to a variable global selection intensity, and thus to a variable takeover time. Indeed, if we keep the shape of the grid constant (say a square) but we allow the cell update mode to change, we observe such an effect on the selection pressure. In fact, it is found that the global selection pressures induced by the various asynchronous policies fall between the low synchronous limit and the high panmictic bound (see Fig. 5.2 and [62]). Thus, by varying the update policy it is possible to influence the explorative or exploitative character of the search.

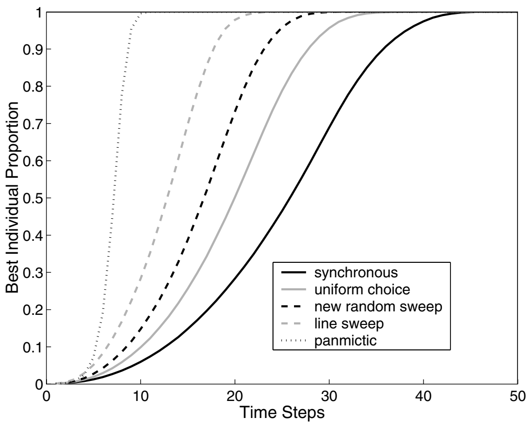


Fig. 5.2. Takeover times with tournament selection in a 32×32 grid with a von Neumann neighborhood. Mean values over 100 runs. The vertical axis represents the fraction of the population consisting of the best individual as a function of time

Turning now to the effects of the grid shape, an interesting result is that algorithms with a similar ratio show a similar selection pressure, as stated in [127]. In Fig. 5.3 we can see growth curves for two algorithms with different neighborhood and population radii, but with similar ratio values. The cases plotted are those of a 32×32 population with a von Neumann neighborhood, and a population of size 64×64 with a Compact21 (C21) neighborhood. In a C21 neighborhood, a central cell is surrounded by two cells in all directions and the four corner cells are cut out.

Now, to observe how the shape of the grid influences the induced selection pressure of the algorithm, the growth curves for several different cGAs using the von Neumann neighborhood and six possible grid shapes are plotted in Fig. 5.4 for a population of 1024 individuals. Note that the selection pressures induced in synchronous rectangular grids is lower than that indicated by the curve for a synchronous square grid (32×32 population), which, as we found before, means that thinner grids favor a more explorative style of search. This figure can be compared with Fig. 4.17, in which a linear scale is used for time.

5.4 Test Suite

Choosing a set of test problems for benchmarking purposes is always a risky exercise, a fact that was already noted in Chap. 3. Indeed, no test suite can

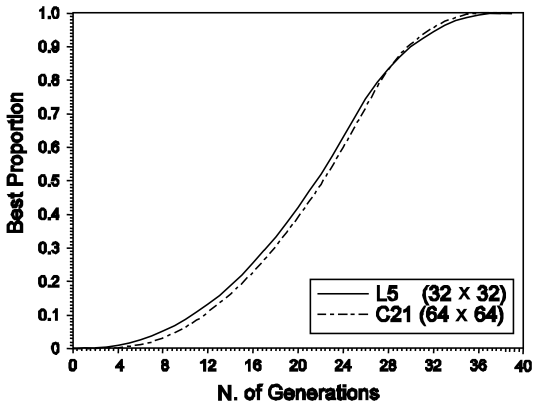


Fig. 5.3. Growth curves of the best individual for two cGAs with different neighborhood and population shapes, but similar ratio values. The vertical axis represents the proportion of population consisting of best individuals as a function of time. Selection is by binary tournament in both cases

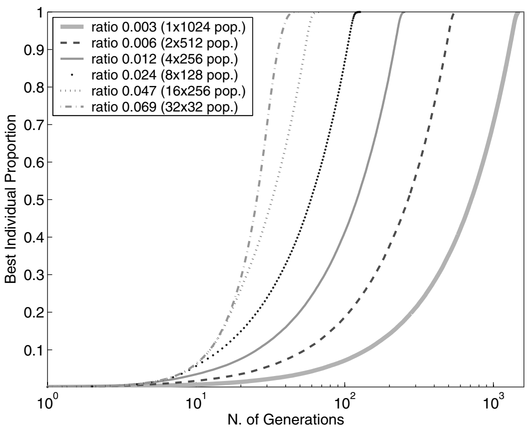


Fig. 5.4. Takeover times with binary tournament selection in populations of 1024 individuals with different grid shapes and a von Neumann neighborhood. Mean values over 100 runs. The vertical axis represents the proportion of the best individual in the population as a function of time. The horizontal axis is on a logarithmic scale

represent all the kinds of problems that are likely to arise in practice. Moreover, even if this was possible, the no-free-lunch (NFL) theorem, or theorems, [160] tell us that, averaged over all problems and problem instances, the performance of all search algorithms is the same. In spite of this, people normally want to solve specific problems for which there is extra information available. This problem-specific knowledge can thus be used to improve the search, since the NFL theorem does not prevent one finding an excellent searcher for a given problem or problem family; all that it says is that this same algorithm will be bad on some other problems. Therefore, although no algorithm can be said to be superior in all cases, there is still scope for algorithms that behave well on particular problems or problem classes.

Incidentally, we can also see that the NFL theorems do not rule out the possibility of constructing a *portfolio* of searchers, each of which, while not optimal, is at least particularly good for a given family of problems or problem instances, or problem subspace. Of course, how to efficiently organize the set of algorithms in order for them to search effectively as a function of easily computed and significant fitness landscape indices or of the progress of the search, is an open problem. An original approach is suggested in [81].

Another important consideration is that the tests in this chapter have only an illustrative purpose. The cEA used, a straightforward genetic cellular algorithm, has not been tuned, nor does it include local search capabilities or problem knowledge other than what is contained in the individual representation and the fitness function. Of course, such improvements would be needed if the algorithms were intended to compete with the best solvers for a given problem or problem class. Here we are mainly interested in comparing cEAs among themselves as a function of two important parameters that have been dealt with at length in the last and present chapters: operation timing and grid shape.

That said, this benchmark is rather representative because it contains several important features found in optimization, such as epistasis, multimodality, deceptiveness, and problem generators. Moreover, in order to be more complete, both combinatorial (discrete) optimization problems and continuous ones are used. These are important ingredients in any work that tries to evaluate algorithmic approaches with the objective of obtaining reliable results, as stated by Whitley et al. in [158]. For convenience of presentation, I shall discuss the two main problem classes (discrete and continuous) separately. Let us begin with the combinatorial problems.

5.4.1 Discrete Optimization Problems

The problems used here are the massively multimodal deceptive problem (MMDP), the multimodal problem generator P-PEAKS, the error-correcting-code (ECC) design problem, and the problem of maximum cut of a graph (MAXCUT). Although there cannot be an optimal choice, as explained above, this set of problems seem at least to be rather representative of different degrees of difficulty and of various important application domains. Given the computational limitations that any experiment must face, this should be enough for us to obtain a good level of confidence in the results.

The problems are briefly described below in order to make the discussion self-contained, as far as possible.

Massively Multimodal Deceptive Problem (MMDP)

The MMDP is a problem that has been specifically designed to be difficult for an EA [68]. It is made up of k deceptive subproblems (s_i) of 6 bits each, whose values depend on the number of ones (*unitation*) in a binary string (see Fig. 5.5). It is easy to see that these subfunctions have two global maxima and a deceptive attractor at the midpoint.

Unitation	Subfunction value
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

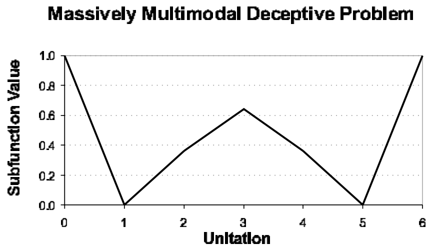


Fig. 5.5. Basic deceptive bipolar function (s_i) for MMDP

In the MMDP, each subproblem s_i contributes to the fitness value according to its unitation(Fig. 5.5). The global optimum has a value of k and is attained when every subproblem is composed of zeros or six ones. The number of local optima is quite large (22^k), while there are only 2^k global solutions. Therefore, the degree of multimodality is regulated by the parameter k. We use here a considerably large problem instance with $k = 40$ subproblems. The instance we try to maximize for solving the problem is shown in the following equation, and its maximum value is equal to k:

$$f_{MMDP}(\mathbf{s}) = \sum_{i=1}^{k} fitness_{s_i}.$$

Multimodal Problem Generator (P-PEAKS)

The P-PEAKS problem [84] is a multimodal problem generator. A problem generator is an easily parameterizable task which has a tunable degree of epistasis, thus allowing one to derive instances of increasing difficulty at will. Also, using a problem generator removes the opportunity to hand-tune algorithms to a particular problem, therefore allowing more fairness when comparing algorithms. With a problem generator, the algorithms are run on a high number of random problem instances, since a different instance is solved each time the algorithm runs, the predictive power of the results for the problem class as a whole is increased.

The idea of P-PEAKS is to generate P random N-bit strings that represent the location of P peaks in the search space. The fitness value of a string is the number of bits that the string has in common with the nearest peak in that space, divided by N (as shown in 5.3). By using a small/large number of peaks we can obtain weakly/strongly epistatic problems. In the work described here we have used an instance of $P = 100$ peaks of length $N = 100$ bits each, which represents a medium to high epistasis level [6]. The maximum fitness value for this problem is 1.0. The fitness value is given by

$$f_{P-PEAKS}(\mathbf{x}) = \frac{1}{N} \max_{1 \le i \le p} \{N - HammingD(\mathbf{x}, Peak_i)\}. \tag{5.3}$$

Error-Correcting-Code Design Problem (ECC)

The ECC problem was presented in [95]. We shall consider a three-tuple (n, M, d), where n is the length of each codeword (number of bits), M is the number of codewords, and d is the minimum Hamming distance between any pair of codewords. The objective is to find a code which has a value of d as large as possible (reflecting greater tolerance to noise and errors), given previously fixed values of n and M. The problem studied here is a simplified version of that in [95]. In our case, we search half of the codewords ($M/2$) that will make up the code, and the other half is made up by the complement of the codewords computed by the algorithm.

The fitness function to be maximized is

$$f_{ECC} = \frac{1}{\displaystyle\sum_{i=1}^{M} \sum_{j=1, i \ne j}^{M} d_{ij}^{-2}},$$

where d_{ij} represents the Hamming distance between codewords i and j in the code C (made up of M codewords, each of length n). We consider here an instance where $M = 24$ and $n = 12$. The search space is of size $\binom{4096}{24}$, which is approximately 10^{87}. The optimum solution for $M = 24$ and $n = 12$ has a fitness value of 0.0674 [30].

Maximum Cut of a Graph (MAXCUT)

The MAXCUT problem looks for a partition of the set of vertices (V) of a weighted graph $G = (V, E)$ into two disjoint subsets V_0 and V_1 such that the sum of the weights of the edges with one endpoint in V_0 and the other one in V_1 is maximized. Individuals are encoded as binary strings $(x_1, x_2, \ldots, x_n)$ of length n, where each digit corresponds to a vertex. If a digit is 1 then the corresponding vertex is in the set V_1; if it is 0 then the corresponding vertex is in the set V_0. The function to be maximized [87] is

$$f_{MAXCUT}(\mathbf{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \cdot \left[x_i \cdot (1 - x_j) + x_j \cdot (1 - x_i)\right]$$

Note that w_{ij} contributes to the sum only if nodes i and j are in different partitions. While one can generate random instances of a graph to test the algorithm, here we have used the case "cut20.09", with 20 vertices and a probability 0.9 of having an edge between any two randomly chosen vertices. The maximum fitness value for this instance is 56.740064.

5.4.2 Experimental Analysis

Although a full-length study of the problems presented in the previous subsection is beyond our scope, I shall present results comparing synchronous and asynchronous cGAs, and also cGAs that have different values of the ratio but the same neighborhood shape (von Neumann). Again, note that the aim here is not to compare the performance of cGAs with state-of-the art algorithms and heuristics for combinatorial and numerical optimization. To achieve this, at least the cGA parameters would have to be tuned and local search capabilities would have to be built into the algorithm, which is not the case here. Thus, the results pertain only to the performance of the different cGA update methods, with different ratios, relative to each other.

The results were obtained using JCell v1.0, a custom simulation program written in Java, with three different static ratios, and with the four asynchronous update modes previously described. The cGAs based on these ratios were synchronous. The configuration of the algorithm is detailed in Table 5.1, while the static ratios used are shown in Table 5.2.

Population size	400 individuals
Selection of parents	Binary tournament + binary tournament
Recombination	DPX, $p_c = 1.0$
Bit mutation	Bit-flip, $p_m = 1/L$
Length of individual	L
Replacement	Rep_if_Better

Table 5.1. Parameterization used in the algorithm

Name	(shape of population)	Value of ratio
Square	(20 × 20 individuals)	0.11
Rectangular	(10 × 40 individuals)	0.075
Narrow	(4 × 100 individuals)	0.031

Table 5.2. Ratios studied

The following tables show the results for the problem suite: MMDP, Table 5.3; P-PEAKS, Table 5.4; ECC, Table 5.5; and MAXCUT, Table 5.6. One hundred independent runs were performed for each algorithm and for every problem in the test suite. The tables report the average of the final best fitness over all runs, the average number of time steps needed to obtain the optimum value (if obtained), and the hit rate (percentage of successful runs). Therefore, the final distance from the optimum (especially interesting when the optimum is not found), the effort expended by the algorithm, and its expected efficacy, respectively, are reported.

Algorithm	Avg. Solution (best=20)	Avg. Generations	Hit Rate
Square	19.813	214.18	57%
Rectangular	19.824	236.10	58%
Narrow	19.842	299.67	61%
LS	19.518	343.52	23%
FRS	19.601	209.94	31%
NRS	19.536	152.93	28%
UC	19.615	295.72	36%

Table 5.3. MMDP problem with a maximum of 1000 generations

Algorithm	Avg. Solution (best=1)	Avg. Generations	Hit Rate
Square	1.0	51.84	100%
Rectangular	1.0	50.43	100%
Narrow	1.0	53.94	100%
LS	1.0	34.75	100%
FRS	1.0	38.39	100%
NRS	1.0	38.78	100%
UC	1.0	40.14	100%

Table 5.4. P-PEAKS problem with a maximum of 100 generations

Algorithm	Avg. Solution (best=0.0674)	Avg. Generations	Hit Rate
Square	0.0670	93.92	85%
Rectangular	0.0671	93.35	88%
Narrow	0.0673	104.16	94%
LS	0.0672	79.66	89%
FRS	0.0672	82.38	90%
NRS	0.0672	79.46	89%
UC	0.0671	87.27	86%

Table 5.5. ECC problem with a maximum of 500 generations

From inspection of these tables some conclusions can be clearly drawn. First, the asynchronous algorithms tend to need a smaller number of generations to

Algorithm	Avg. Solution (best=56.74)	Avg. Generations	Hit Rate
Square	56.74	11.26	100%
Rectangular	56.74	11.03	100%
Narrow	56.74	11.88	100%
LS	56.74	9.46	100%
FRS	56.74	9.69	100%
NRS	56.74	9.55	100%
UC	56.74	9.58	100%

Table 5.6. MAXCUT problem with a maximum of 100 generations

locate an optimum than do the synchronous ones, except in the case of the MMDP problem. Statistical tests not shown here (see [41]) confirm that the differences between the asynchronous and synchronous algorithms are significant. This indicates that the asynchronous versions perform more efficiently with respect to cEAs with different static ratios, a result that confirms the influence of the more intense selection of asynchronous cEAs.

Conversely, if we pay attention to the success (hit) rate, it can be concluded that the synchronous policies with various ratios outperform the asynchronous algorithms (except for the ECC problems): slightly in terms of the average final fitness, and clearly in terms of the probability of finding a solution (i.e. the frequency of location of the optimum).

Another interesting result is the fact that we can define two classes of problems: those solved by all methods to optimality (100% hit rate) and those in which no 100% rate is achieved at all. The former seem to be suitable for straight cEAs, while the latter need some help, for example by including local search.

In order to summarize the large set of results and draw some useful conclusions, a final ranking of the algorithms following three different metricsis presented: average best final solution, average number of generations for success, and hit rate. Table 5.7 shows the three rankings, which go from 1 (best) to 7 (worst) according to the three criteria.

Avg. solution		Avg. generations		Hit rate	
1 Narrow	4	1 NRS	8	1 Narrow	4
2 Rectangular	9	2 LS	10	2 Rectangular	9
2 FRS	9	3 FRS	11	2 FRS	9
4 NRS	10	4 UC	16	4 NRS	11
5 UC	11	5 Rectangular	19	5 Square	12
5 LS	11	6 Square	21	5 UC	12
7 Square	12	7 Narrow	27	5 LS	12

Table 5.7. Ranking of the algorithms

As one would expect after the previous comments, synchronous algorithms with “narrow” and “rectangular” ratios are in general more accurate than all the asynchronous algorithms, according to the criteria of average best final fitness and of hit ratio, at least for the test problems used here, with a special leading position for narrow population grids. On the other hand, the asynchronous versions clearly outperform any of the synchronous algorithms in terms of the average number of generations, with a trend towards NRS as the best-ranked flavor of cGA for the test suite.

In summary, asynchronous algorithms seem to be numerically more efficient (faster) than synchronous ones for the P-PEAKS, ECC, and MAXCUT problems, but not for the MMDP. On the other hand, synchronous algorithms outperform asynchronous ones in terms of the hit rate for these benchmarks, which could be an important issue for many applications. In particular, the more explorative character of the narrow population structure, with its correspondingly lower selection pressure, seems to allow a more accurate search in most cases. Again, it has to be pointed out that the results cannot be immediately generalized to other problems or problem types. However, the picture that emerges from this empirical investigation is a coherent one, and it essentially confirms the importance of selection intensity considerations.

5.4.3 Continuous Optimization Problems

In this and the next subsection, the empirical tests on combinatorial optimization problems are extended to some continuous functions. The functions that were selected for the study are three typical multimodal numerical benchmarks: Rastrigin’s (RASTR) and Ackley’s (ACKL) functions, and a fractal (FRAC) function. In contrast to the binary-coded problems above, here real coding is used. There are well-known reasons for this choice, a good discussion of which can be found in Chap. 5 of Michalewicz’s book [99]; the floating-point implementation described in that book was used here.

Rastrigin’s Function (RASTR)

The generalized Rastrigin function is a sinusoidally modulated function with a global minimum of zero at the origin. It is a typical example of a nonlinear multimodal function. It was first proposed by Rastrigin as a two-dimensional function and has been generalized by Mühlenbein and Schierkamp-Voosen in [107]. This function is fairly difficult owing to its large search space and its large number of local minima, although the function is separable and the local minima are symmetrical:

$$f_{RASTR}(\mathbf{x}) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(\omega x_i).$$

The constants are given by $A = 10$ and $\omega = 2\pi$. The domain of variables x_i, $i = 1, \ldots, n$, is $-5.12 \le x_i \le 5.12$, and $n = 10$. The function has a global minimum at the point $f(\mathbf{0}) = 0$.

Ackley's Function (ACKL)

Ackley's function is a multimodal test function obtained by cosine modulation of an exponential function. Originally proposed by Ackley [1] as a two-dimensional function, it has been generalized by Bäck et al. [15]. Unlike Rastrigin's function, Ackley's function is not separable, even though it also shows a regular arrangement of the local optima. The function is defined by

$$f_{ACKL}(\mathbf{x}) = -a \exp\left[-b\left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2}\right] - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + e.$$

The constants are $a = 20$, $b = 0.2$, and $c = 2\pi$. The variables $x_i,\ i = 1, \ldots, n$ are in the domain $-32.768 \leq x_i \leq 32.768$. This function has a global minimum at the point $f(\mathbf{0}) = 0$.

Fractal Function (FRAC)

This function has been taken from [14], where its construction, as well as the motivations for introducing it as a test problem, are given. The function allows the degree of ruggedness to be controlled, and it is likely to capture features of real-world noisy objective functions. The function is defined as follows

$$f_{FRAC}(\mathbf{x}) = \sum_{i=1}^{n}(C'(x_i) + x_i^2 - 1),$$

where

$$C'(z) = \begin{cases} \frac{C(z)}{C(1)|z|^{2-D}} & \text{if } z \neq 0, \\ 1 & \text{if } z = 0, \end{cases}$$

and

$$C(z) = \sum_{j=-\infty}^{\infty} \frac{1 - \cos(b^j z)}{b^{(2-D)j}}.$$

The constants were $D = 1.85$ and $b = 1.5$ in the experiments. The 20 variables x_i $(i = 1, \ldots, 20)$ varied in the range $[-5, 5]$. The infinite sum in the function $C(z)$ was calculated in practice by starting with $j = 0$ and alternating the signs of the j values. The sum was stopped when the relative difference between its previous and present values was lower than 10^{-8} or when $j = 100$ was reached.

5.4.4 Experimental Analysis

For the study described here, we use the same asynchronous update policies and grid-axei ratios as were employed for the combinatorial problems. On the other hand, since the representation of the individual was now of floating-point

Population size	400 individuals
Selection of parents	Binary tournament + binary tournament
Recombination	AX, $p_c = 1.0$
Bit mutation	Uniform, $p_m = 1/2L$
Length of individual	L
Replacement	Rep_if_Better

Table 5.8. Parameterization used in the algorithm for the real-encoded problems. "AX" stands for "standard arithmetic crossover"

type, specific genetic operators were needed. The operators and parameters used are detailed in Table 5.8 (see also [99]).

The results of the experiments are reported in the following tables: RASTR, Table 5.9; ACKL, Table 5.10; and FRAC, Table 5.11. As in the discrete problems, these tables contain values for the average of the final best fitness, the average number of generations needed for finding it, and the hit rate. These three values were calculated over 100 independent runs. For these three real-coded problems, a run was stopped successfully as soon as an individual was found with a fitness within 0.1 of the optimum.

Algorithm	Avg. solution (best$\leq$ 0.1)	Avg. generations	Hit rate
Square	0.0900	323.8	100%
Rectangular	0.0883	309.8	100%
Narrow	0.0855	354.2	100%
LS	0.0899	280.9	100%
FRS	0.0900	289.6	100%
NRS	0.0906	292.2	100%
UC	0.0892	292.4	100%

Table 5.9. RASTR problem with a maximum of 700 generations

Algorithm	Avg. solution (best$\leq$ 0.1)	Avg. generations	Hit rate
Square	0.0999	321.7	78%
Rectangular	0.0994	293.1	73%
Narrow	0.1037	271.9	65%
LS	0.0932	302.0	84%
FRS	0.0935	350.6	92%
NRS	0.0956	335.5	87%
UC	0.0968	335.0	85%

Table 5.10. ACKL problem with a maximum of 500 generations

The results obtained on continuous problems are not as clear as in the discrete case. With respect to the average number of generations needed to

Algorithm	Avg. solution (best$\leq$ 0.1)	Avg. generations	Hit rate
Square	0.0224	75.2	94%
Rectangular	0.0359	62.8	78%
Narrow	0.1648	14.6	16%
LS	0.0168	69.7	98%
FRS	0.0151	71.5	100%
NRS	0.0163	73.6	98%
UC	0.0138	72.8	96%

Table 5.11. FRAC problem with a maximum of 100 generations

find an optimal solution, the asynchronous algorithms are not always faster than the synchronous ones; examples are the ACKL and FRAC problems, but the differences are not always statistically significant. Also, unlike what was observed in the case of discrete problems, the success rates of the asynchronous algorithms are often higher than those of the synchronous ones. The FRS cEA is the only algorithm which is able to find the solution in all executions for the FRAC problem.

These results show that although the search behavior of a cEA is certainly related to its induced selection pressure, several other factors enter into the global picture. The actual behavior results from the interplay of selection, representation, and the particular genetic operators used and their parameters. And, of course, we should not forget that, in the end, the nature of the particular fitness landscape is the single most important feature of the problem that influences the search.

In order to summarize these results, Table 5.12 gives a ranking of the algorithms for all the problems in terms of the average solution found, the number of generations needed to find an optimal solution, and the success rate. It can be seen from this table that there exists a trend for asynchronous algorithms to perform better than synchronous ones in terms of the average solution found and the success rate, while synchronous algorithms with a variable ratio seem to be more efficient than asynchronous algorithms in general (The "square" and "LS" algorithms are the exceptions).

Avg. solution		Avg. generations		Hit rate	
1	8	1 LS	7	1 FRS	3
2 LS	9	2 Narrow	9	5 NRS	5
2 FRS	9	2 Rectangular	9	4 LS	7
4 NRS	13	4 FRS	13	6 UC	8
4 Rectangular	13	5 UC	14	7 Square	11
6 Narrow	15	6 NRS	15	3 Rectangular	13
7 Square	16	7 Square	17	2 Narrow	15

Table 5.12. Ranking of the algorithms with continuous problems

5.5 Cellular Genetic Programming

To pursue the empirical analysis, in this section I present another experimental investigation of cellular evolutionary algorithms, this time based on genetic programming. This case study uses the same test functions that were described in Sect. 3.2, to which the reader is referred for an introduction. In Chap. 3 we studied the empirical behavior of multiple, communicating GP populations i.e. island GP, for these problems. In this section we shall compare this population structure with a cellular population of the same size on a square two-dimensional toroidal lattice, on the same benchmark problems.

While cGAs or cellular cvolution strategies (cESs) are rather common, cellular genetic programming (cGP) has rarely been used. However, there are no particular reasons for this lack of interest, and cGP is certainly a worthy member of the evolutionary-computing family, as some investigations have clearly shown [55, 139].

For the experiments described here, the parameters were the same as those of Chap. 3 for standard genetic programming and island GP. For cGP, a two-dimensional square grid with periodic boundary conditions and a von Neumann neighborhood was used. The algorithm was synchronous, selection was done by binary tournament, the crossover and mutation were the same as in the "standard" version of GP (see Sect. 3.2), and the central individual was replaced by the best individual in the neighborhood after the genetic operations.

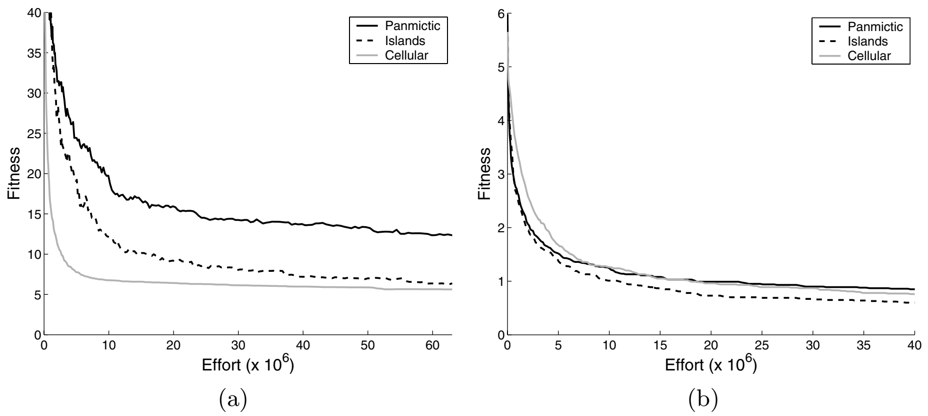
5.5.1 Experimental Results

I shall show a summary of the results in two areas, as in Chap. 3: computational effort versus time, and diversity. Details of the implementation and more data can be found in [57].

5.5.2 Fitness Evolution

The averages of the best fitness against effort are reported in Fig. 5.6 for the three test problems and for the three population structures: panmictic, islands, and toroidal grid. Again, we can see how the "magic of structured population" manages to improve results in a significant manner. Indeed, both cGP and island GP find better solutions quicker than standard GP on the average. And this is confirmed by statistical significance tests (see [57]).

On the other hand, cGP and island GP are not easy to tell apart statistically and have roughly similar problem-solving performances. For the artificial ant problem (Fig. 5.6(a)) the cellular model outperforms the island model, requiring less effort to reach the same average solution fitness level. For the even parity and symbolic regression problems (Fig. 5.6(b) and (c)), cGP shows a quick improvement of fitness at the beginning of the runs, but then the curve levels-off and the island system curve crosses it over and converges to better



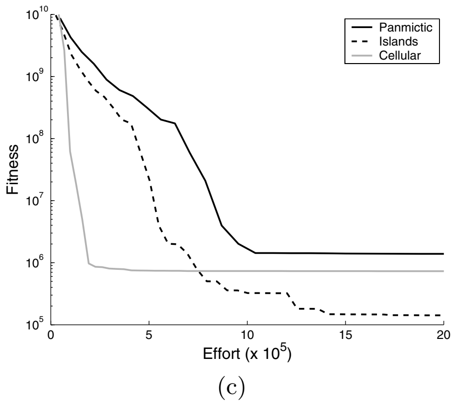
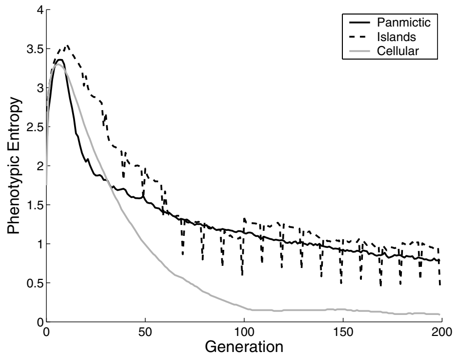


Fig. 5.6. Best mean fitness vs. computational effort. Ant problem (a), even-parity-4 problem (b), and symbolic regression problem (c) (note the logarithmic scale on the abscissae in (c)). The curves are averages over 100 independent runs

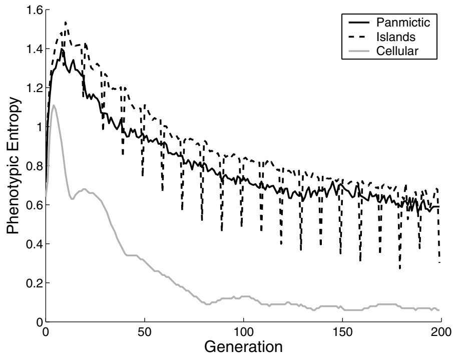
average solution quality. This behavior is related to these particular problems and to exploitation/exploration characteristics of the algorithms, as we have seen. The issue will taken up again below when analyzing the diversity evolution behavior of the algorithms, a feature that is directly related to exploration/exploitation character. For even harder test problems it might pay to be less exploitative i.e., cGP might have an edge in these cases.

5.5.3 Diversity Evolution

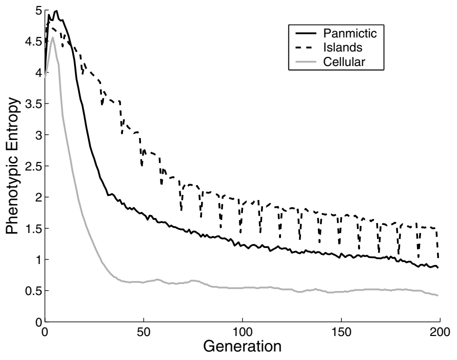
In the following paragraphs the evolution of diversity is studied for multipopulation and cellular genetic programming. For the sake of comparison, the results for standard GP are also reported in the figures. The various diversity measures have been defined in Sect. 3.9 of Chap. 3. Apart from those, for cGP, and any cEA, it is possible to define other statistical measures that are related to diversity. These statistics take into account the local, slow diffusion nature of cEAs and are certainly more relevant at the cell assembly level. For brevity these will not be dealt with here, but the interested reader can find the relevant definitions in [27], and their application to cGP in [57].



(a)



(b)



(c)

Fig. 5.7. Phenotypic entropy against generation number. Ant problem (a), even 4 parity problem (b), and symbolic regression problem (c) Curves are averages over 100 independent runs.

Phenotypic Diversity

We first discuss the phenotypic behavior. Figure 5.7 shows the phenotypic entropy for the three test problems. Entropy represents the amount of disorder of the population, thus low entropy means low diversity. However, since the phenotypic measure compares the number of different fitness values, it could be interpreted as the number of groups having the same fitness value.

Thus high entropy can be considered as the presence in the population of a high number of small groups of individuals, each group having the same fitness value, while low entropy would mean a low number of large groups of individuals.

In this perspective, the fact that the cellular model always has a lower phenotypic entropy with respect to both the island and the panmictic models, as can be seen in the figure, can be interpreted as the presence in the population of a low number of groups each containing many individuals having the same fitness value. This is confirmed by the low phenotypic variance of the cellular model shown in Fig. 5.8. The jigged behavior of the curves referring to the subpopulations in the island model is due to the sudden change in diversity when the new individuals enter the population at fixed generation numbers, since the migration model is synchronous.

Low phenotypic diversity in the cellular model can be explained by the slow diffusion of the information across the grid that induces patches of individuals having similar characteristics. It is worth to point out that low phenotypic entropy does not imply worst problem-solving capabilities for cGP. Indeed, looking back at the curves in Fig. 5.6 clearly shows that the quality of the average solutions found, as well as the effort that is spent are comparable to those of the island model, as already noted in the previous section.

Genotypic Diversity

As in the case of phenotypic entropy, Fig. 5.9 shows that genotypic entropy is lower for the cellular model with respect to both the island and the panmictic ones, while genotypic entropy for the island model is almost the same as for the panmictic model for ant and parity problems, and lower for symbolic regression.

This behavior suggests that, as regards the cellular model, there are few groups of individuals having the same distance from the empty tree, each group being composed by many trees. However, as Fig. 5.10 suggests, trees in the population are very dissimilar among them because the variance is high, thus the distance of each tree from the origin tree is substantially different from the average distance of all the trees from the origin tree.

The coexistence of high genotypic diversity and low phenotypic diversity in the cellular model could seem contradictory. However this apparent conflicting behavior can be explained by the fact that though the trees are structurally different, this does not imply that their fitness must be different too. In the cellular case it means that almost all the trees have good fitness values and this explains the good convergence of the cellular model. In addition, we know that the same overall behavior is observed in GP for the test problems irrespective of the population structure. The only important difference is that in cGP the diversity is strongly related to the geographical position in the grid, a situation that cannot be detected from the global diversity measures but that is clearly apparent from local statistics (see [27, 57]).

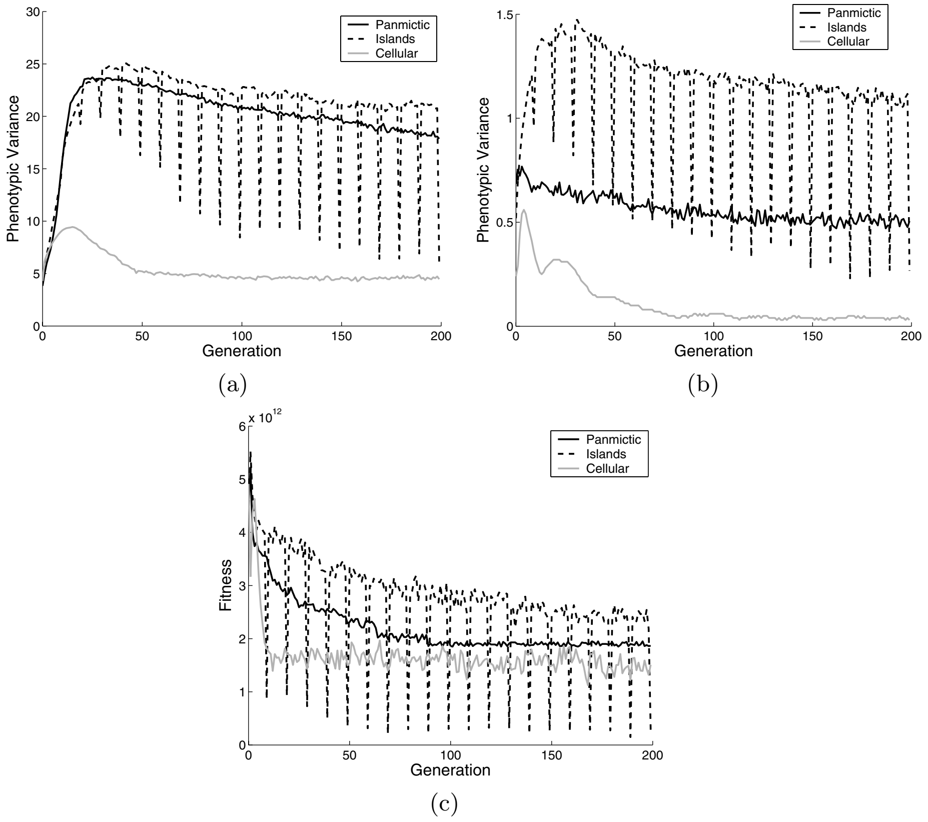
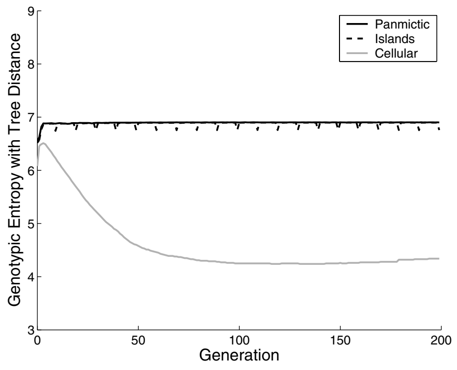


Fig. 5.8. Phenotypic variance against generation number. Ant problem (a), even 4 parity problem (b), and symbolic regression problem (c) Curves are averages over 100 independent runs.

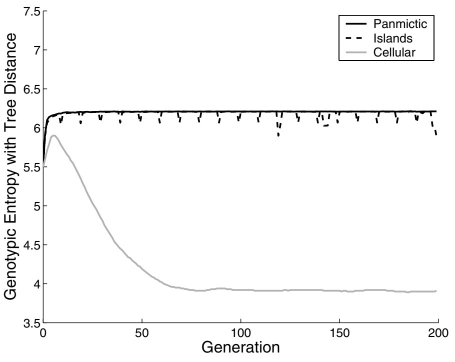
5.6 Summary

In this chapter a rather systematic exploration of the empirical properties of cEAs has been presented. The results on the test problems for cGAs confirm, with some exceptions, that the problem-solving capabilities using the various update/ratio modes are correlated to their induced selection pressures, showing that exploitation plays an important role. It is clear that the role of exploration might be more important on even harder problem instances, but this aspect can be addressed in the algorithms by using more explorative settings, as well as by using different cEA strategies at different times during the search dynamically. This last idea has been put to practice in [4] where the grid ratio is changed automatically in a self-adaptive manner.

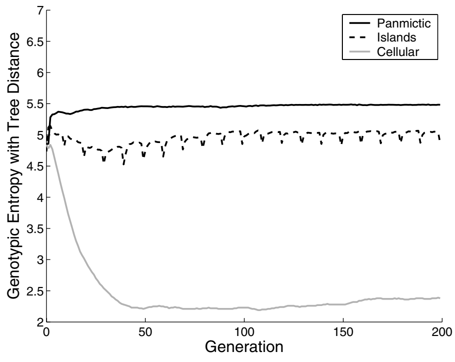
For the discrete optimization problems asynchronous algorithms are generally faster than synchronous ones for most problems, as expected from the



(a)



(b)



(c)

Fig. 5.9. Genotypic entropy against generation number. Ant problem (a), even 4 parity problem (b), and symbolic regression problem (c) Curves are averages over 100 independent runs.

respective selection pressures. On the other hand, synchronous algorithms outperform asynchronous ones in terms of the hit rate for the problems studied. This trend is less clear in the continuous problems, where the differences among the algorithms are smaller. Flatter grids further favor exploration over exploitation. In conclusion, cEAs seem to possess a number of natural degrees of freedom that can be easily tuned as a function of the problem, thus offering a richer range of behaviors than panmictic algorithms.

Finally, the experiments on the seldom used synchronous cGP have shown that it is more efficient than standard GP and roughly equivalent to multi-population GP for the problems studied.

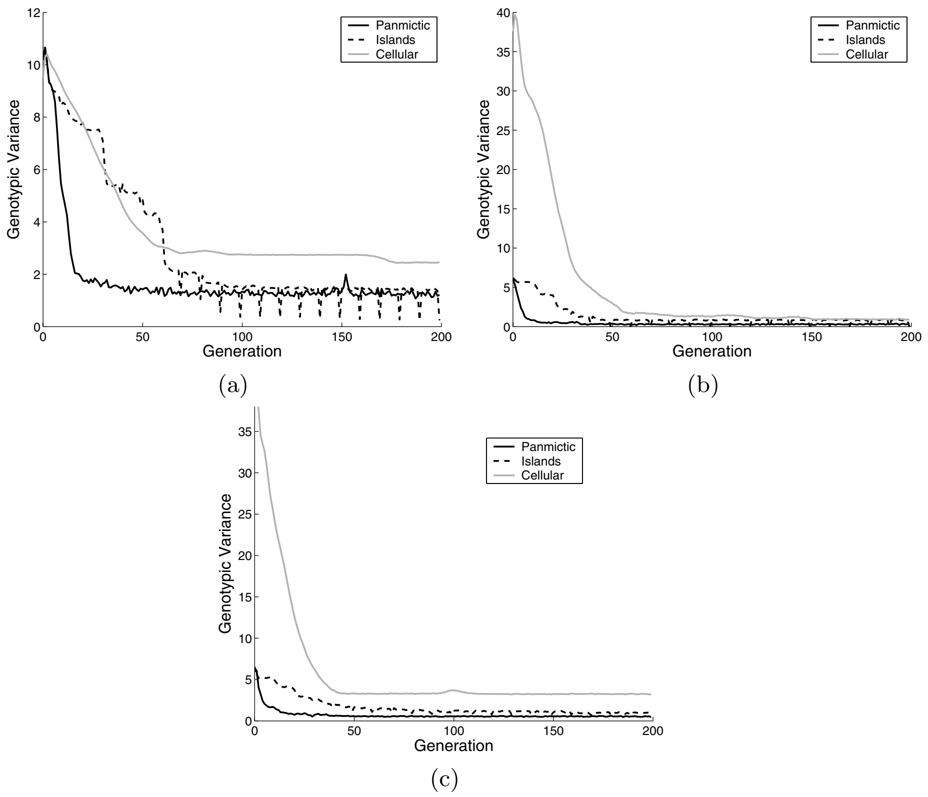


Fig. 5.10. Genotypic variance against generation number. Ant problem (a), even 4 parity problem (b), and symbolic regression problem (c) Curves are averages over 100 independent runs.

6

Random and Irregular Cellular Populations

In Chap. 3, I described and analyzed the replication and propagation of individuals under the effects of selection in cellular populations possessing a regular lattice structure. This kind of topology is a natural one, and it has often been used in the theory and applications of cEAs. We have seen that the lattice topology has a strong influence on the selection pressure, and thus on the explorative or exploitative character of the search. However, regular lattices are just a class of possible structures. Nothing prevents us from giving the population any graph structure that we please. The most general graph structure is, in a sense, the random graph, to be defined below. We shall thus investigate the similarities the differences between panmictic populations and cellular populations that are randomly structured, in the first part of this chapter. But the story does not end here. The last few years have seen a dramatic increase of interest in the structure of the big networks that form part of our daily environment such as the World Wide Web, the Internet, electrical power networks, and many others. These networks have properties that are unparalleled in lattices or random networks. In turn, the structure of these networks gives rise to a wide range of dynamical behaviors. We shall thus examine the influence that these kinds of topologies may have on cellular evolutionary algorithms in the second part of the chapter. The reader is warned that these kinds of population topologies are seldom, if ever, used in EA work. In spite of this, I have chosen to include them for a number of reasons, some of which will become clear further ahead in the chapter.

The following section is an introduction to random graphs to an extent that, though brief, should be sufficient to understand the subject matter of this chapter. The reader is advised at this point to go back to Sect. 1.1 to find the relevant concepts and definitions related to graphs.

6.1 Random Graphs

The random-graph model was formally defined by Erdös and Rényi at the end of the 1950s [21]. In its simplest form, the model consists of N vertices joined by edges that are placed between pairs of vertices uniformly at random. In other words, each of the possible $N(N-1)/2$ edges is present with probability p and absent with probability $1-p$. The model is often called $G_{N,p}$ to point out the fact that, rigorously speaking, there is no such thing as a random graph, but rather an ensemble $G_{N,p}$ of equiprobable graphs.

Another closely related model of a random graph considers the family of graphs $G_{N,M}$ with N vertices and exactly M edges. For $0 \leq M \leq \binom{N}{2}$, there are $s = \binom{N(N-1)/2}{M}$ graphs with M edges. If the probability of selecting any one of them is $1/s$, then the ensemble $G_{N,M}$ is called the family of uniform random graphs. For $M \sim pN$ the two models are very similar, but we shall use $G_{N,p}$ in what follows.

A few simple facts are worth noting about random graphs. The *average degree* $\langle k \rangle$ of a graph G is the average of all the vertex degrees in G: $\langle k \rangle = (1/N)\sum_{j=1}^{N} k_j$, where k_j is the degree of vertex j. If $|E| = M$ is the number of edges in G, then $M = (N\langle k \rangle)/2$, since $\sum_{j=1}^{N} k_j = 2M$ (each edge is counted twice).

The expected number of edges of a random graph belonging to $G_{N,p}$ is clearly $(1/2)N(N-1)p$, but since each edge has two ends, the average number of edge ends is $N(N-1)p$, which in turn means that the average degree of a vertex in a random graph is

$$\langle k \rangle = \frac{N(N-1)p}{N} = (N-1)p \simeq Np, \tag{6.1}$$

for sufficiently large N.

Another important property of a connected random graph is that the average path length is of the order of $\log N$.

The Erdös and Rényi model has a number of interesting properties that are outside the scope of this book (see for example [21]). However, one rather surprising property is worth mentioning: the presence of a critical point at which a *phase transition* takes place. Indeed, Erdös and Rényi proved that for small values of $\langle k \rangle$ there are few edges in the graph, which is globally disconnected, and the connected components are small. Above a critical value of $\langle k \rangle = 1$, however, a large connected component appears, which is called the *giant component* and whose size is proportional to N. Below the critical point, i.e. for $\langle k \rangle < 1$, the typical size of the small components is $O(\log N)$. For the sake of illustration, although it is too small for the statistics to be fully reliable, Fig. 6.1 depicts a random graph with $N = 40$ and $p = 0.1$. The graph is sparse, with an average degree $\langle k \rangle$ of about four, and there is a giant connected component. There are also some unconnected vertices, which are components of size one.

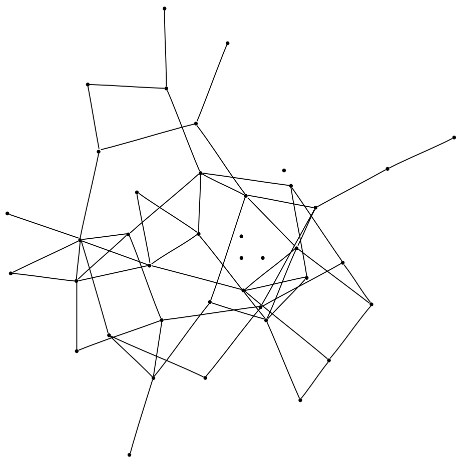


Fig. 6.1. A small random graph with 40 vertices and a probability of connection of two vertices p equal to 0.1

The concepts outlined above will now be used in modeling selection intensity in randomly structured cellular populations.

6.2 Selection Intensity in Random Cellular Populations

The random graph is a well-studied structure in mathematics, biology, and the social sciences. For example, models of infection transmission in a population of individuals with random links between them have been known for years [111]; likewise, information transmission in society and in the economy has sometimes been modeled using a random graph structure. Most propagation models are based on differential equations. Instead, here we use discrete models based on difference equations, which seems more suitable for finite evolving spatial populations.

Equation (4.1) defines the random variable $N(t)$ denoting the number of copies of the best individual in the population at time step t. Below we give the recurrences describing the growth of the random variable $N(t)$ in evolving populations in cEAs structured on graphs for the synchronous and two of the asynchronous update policies described in Chap. 4. In general, such recurrences take the common form

$$E[N(t)] = \sum_{i=1}^{n} P[N(t-1) = i] \left(i + \Delta_i(t-1)\right), \tag{6.2}$$

where $\Delta_i(t-1) = N(t) - N(t-1)$, given $N(t-1) = i$, is a random variable as well. This random variable will depend on the update method. Let us suppose that at time step $t-1$ there are i copies of the best individual. At the next time step, each of the other $n-i$ individuals will contain a copy of the best with a probability depending on its number of neighbors, the number of its neighbors containing a copy of the best and the selection operator. The first two conditions can be seen as random variables, both depending on the topology of the population.

Therefore, in (6.2) we have

$$\varDelta_i(t-1) = \sum_{r=1}^{n-i} \sum_{j=1}^{n-1} P[K=j] \sum_{l=0}^{j} P[B_j = l] p_{\mathrm{sel}}(j,l), \tag{6.3}$$

where K denotes the number of neighbors of an individual, B_j stands for the number of copies of the best individual in a neighborhood of j individuals, and $p_{\mathrm{sel}}(j,l)$ is the probability of selecting a copy of the best individual from the l best of j neighbors.

According to the definitions in Sect. 6.1, random graphs with n vertices belonging to the family $G_{n,q}$ can be constructed by taking all possible pairs of vertices and connecting each pair with probability q[1]. In the general case of a cEA in which a population structured as a random graph with a probability $0 < q < 1$ of having an edge between any pair of vertices evolves, the random variables K and B_j of (6.3) have the following probability functions:

$$P[K=j] = q^j (1-q)^{n-1-j},$$
$$P[B_j = l] = \begin{cases} 1 & \text{if } l = (ij)/(n-1), \\ 0 & \text{otherwise}, \end{cases}$$

since any of the j neighbors of an individual has a probability $i/(n-1)$ of containing a copy of the best individual.

We have seen that cellular evolutionary algorithms are good candidates for using selection methods that are easily extensible to small local pools, such as ranking and tournament. The equations for the growth of individuals for those local selection policies are mathematically rather complicated for randomly structured populations, involving higher moments of the distribution. Therefore, here I use a simplified selection policy, called *uniform selection*, which gives rise to a useful and interesting model.

This selection mechanism randomly selects an individual in the selection pool (i.e. the neighborhood of a given individual). The selected individual then replaces the first individual under consideration if it has a better fitness. Such an operator is similar to the *local parent selection* introduced by Gorges-Schleuter in [72], except that a $(\mu+\lambda/\mu,\nu)$-LES is used instead of a $(\mu,\lambda/\mu,\nu)$-LES, with $\kappa = \infty$ (κ being the upper limit for the life span) and $\rho = 1$ (ρ being the number of selected ancestors).

[1] I use n and q in this section instead of the customary N and p to avoid confusion with $N(t)$ and the probability of selection p.

In a cEA whose population is structured as a random graph, since the number of edges incident on a given edge is a binomial random variable, we can use the *mean-field hypothesis*, which consists in taking for all vertices the average number of neighbors $q(n-1)$. In this way, any vertex “sees” the same isotropic average environment. Under this hypothesis, the expected number of copies of the best individual in a neighborhood not containing a copy of the best at time step t is $E[B_{q(n-1)}] = qN(t)$. We shall see that this approximation is good unless the probability q is very low. In the case of uniform selection, the probability that a copy of the best is selected at time step t is

$$p_{\text{sel}}^{\text{rnd}}(q(n-1), qN(t)) = \frac{qN(t)}{q(n-1)} = \frac{N(t)}{n-1}. \tag{6.4}$$

This approximation, valid when the mean-field hypothesis can be used, gives a probability equal to that of the panmictic case, where the graph describing the topology of the population is a complete graph. In fact, for this structure, each individual has exactly $K = n-1$ neighbors, and the number of copies of the best individual in a neighborhood not containing a copy of the best at time step t is $B_{n-1} = N(t)$. In the case of uniform selection, the probability that a copy of the best is selected at time step t is

$$p_{\text{sel}}^{\text{pan}}(n-1, N(t)) = \frac{N(t)}{n-1}. \tag{6.5}$$

Therefore, the selection pressure for a randomly structured population is similar to that for a panmictic one, when the mean-field hypothesis can be applied.

For synchronous update, using (6.2) and (6.5), the recurrence for $N(t)$ can be written as

$$\begin{cases} N(0) = 1 \\ E[N(t)] = E[N(t-1)] + (n - E[N(t-1)])(E[N(t-1)])/n, \end{cases} \tag{6.6}$$

which is a typical form of a discrete logistic recurrence.

The two asynchronous policies can be treated in a similar manner. Since the derivation is rather cumbersome, the resulting growth equations are omitted here but can be found in [64].

6.3 Experimental Results

We report experimental data for the three update policies and panmictic and randomly structured populations using uniform selection. The synchronous and two asynchronous update modes were used, the latter being new random sweep and uniform choice. See Chap. 4 for the definitions of the asynchronous modes.

In all the curves in Fig. 6.2 the population grows exponentially at first and then saturates, giving the usual sigmoidal shape for the growth curves.

However, one can distinguish the three update policies very clearly, with NRS being faster than the synchronous mode. The UC policy starts in a similar waysimilar to NRS and then joins the synchronous case when saturation sets in.

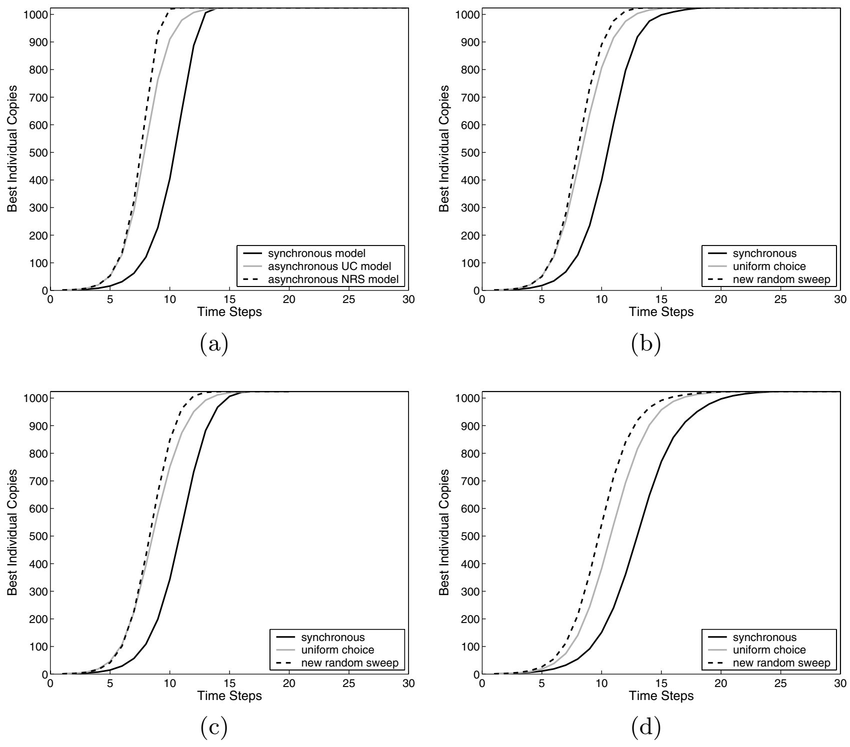


Fig. 6.2. Theoretical growth curves for the three update policies (a). Experimental growth curves for the panmictic case (b), a random graph with $q = 0.1$ (c), and a random graph with $q = 0.01$ (d), for the three update policies. The population size is 1024. The experimental curves in (b), (c), and (d) are averages over 100 independent runs

Figure 6.2 a depicts the theoretical curves corresponding to the three update modes specified above. Of course, according to the mean-field approximation, the panmictic and random-graph cases are actually the same. This is clearly confirmed by Figs. 6.2 b,c, which show the experimental curves for the panmictic case and the random-graph case with probability $q = 0.1$. Figure 6.2 d shows the experimental random-graph case with $q = 0.01$: we observe that for low probabilities, the mean-field hypothesis gives a worse approximation to the experimental results. It should be noted that for low q values there is a nonnegligible probability that the generated random graph is disconnected. To avoid these cases, we considered only connected graphs in our experiments, randomly sampled from the family $G_{n,q}$ of all possible random graphs with n vertices and edge probability q.

Table 6.1 gives the predicted and experimental average takeover times for the three update modes. Results are reported for the panmictic case and the two randomly generated graph structures previously described .

	Synchronous	Asynchronous NRS	Asynchronous UC
Predicted	14	11	16
Panmictic mean	14.84 (1.36)	11.52 (1.25)	16.69 (1.99)
Random (q=0.1) mean	15.07 (1.22)	11.96 (1.25)	16.62 (1.68)
Random (q=0.01) mean	20.32 (3.23)	16.46 (2.78)	21.15 (3.53)

Table 6.1. Predicted takeover times and experimental mean takeover times (with corresponding standard deviations) for the three update methods. The experimental results were obtained over 100 independent runs. The population size was $n = 1024$ in all cases

In Figs. 6.3 and 6.4, a direct comparison is provided by superposing on the same graph the theoretical curves for the synchronous (a), asynchronous NRS (b), and asynchronous UC (c) updates, and ten randomly chosen corresponding experimental curves. This is more informative than a comparison with the average experimental curves since the theoretical result is an expectation curve. Figure 6.3 reports results for the random graph with $q = 0.1$; it is clear that there is a very good agreement between the predictions of the models and the experiments. As stated above, we cannot hope that such an agreement will also hold for random graphs with a very low probability q. In fact, Fig. 6.4 shows that the approximation is much worse. This is understandable qualitatively on the following grounds. In a random graph, the node degree is binomially distributed by construction. Therefore, for 1024 vertices, the average number of neighbors of an individual is about 100 for $q = 0.1$ and about 10 for $q = 0.01$. Thus the standard deviation in the latter case is about 3, while it is about 9 in the former. This means that many nodes will have very few edges in the $q = 0.01$ case, which will slow down the propagation rate of the best individual.

6.4 Small-World Networks

Random graphs are interesting objects as they can be used to establish general quantitative properties that are present in those graphs with very high probability (this is a mathematical statement that will not be further qualified here). In our context of structured EAs, we have seen that randomly structured cellular populations have interesting properties that can be modeled quantitatively, to some extent. They are also a useful model for generating

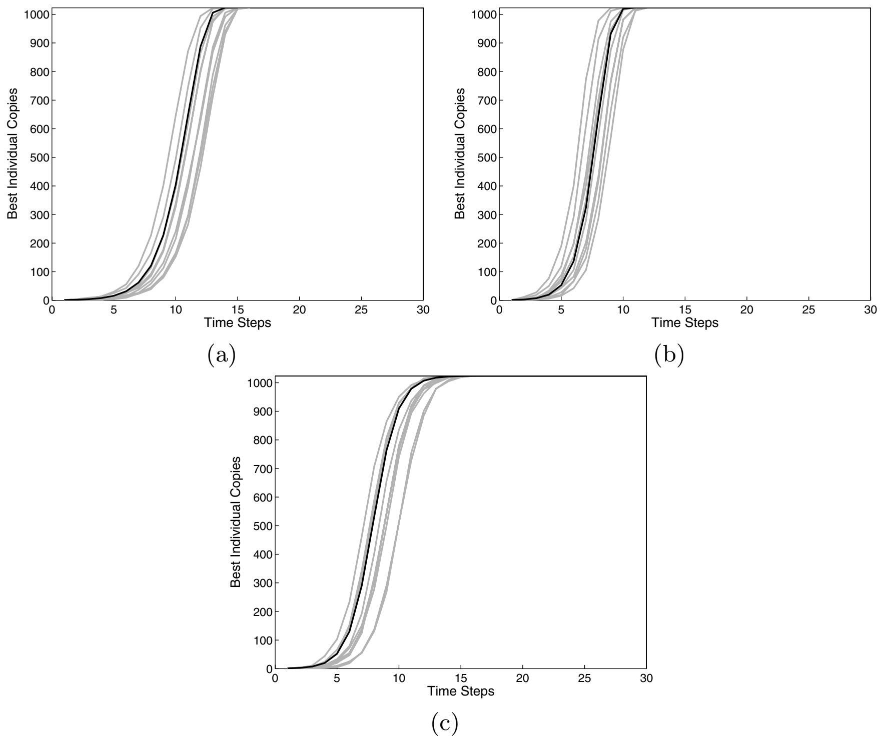


Fig. 6.3. Theoretical curves (black) and experimental curves (gray) for a random-graph population with $q = 0.1$ using synchronous update (a), and asynchronous NRS (b), asynchronous UC (c)

problem instances for testing network algorithms [130], and they are used in other ways too. But are random graphs a useful model of the networks that permeate society? Actually, social scientist felt qualitatively as early as the 1950s that social and professional links and acquaintances did not follow a random structure (see, for instance, Milgrams's experiment in [18]). For example, if a person has some relationship with two others, then the latter two are more likely to know each other than are two arbitrary persons. This does not fit the random-graph model, however, where the likelihood that two given nodes are connected is the same independent of any other consideration.

A quantitative answer to this question came only a few years ago, opening up a flurry of research that is reshaping complex-system thinking to a large extent and is still far from slowing down. In a groundbreaking paper, Watts and Strogatz [155] proposed a network construction algorithm that gives rise to graphs having the following properties: the path length from any node to any other node is short, as in random graphs, but, unlike random graphs, there is local structure in the network. Watts and Strogatz called their networks *small-world networks*, a term that has been in use for a long time in the field

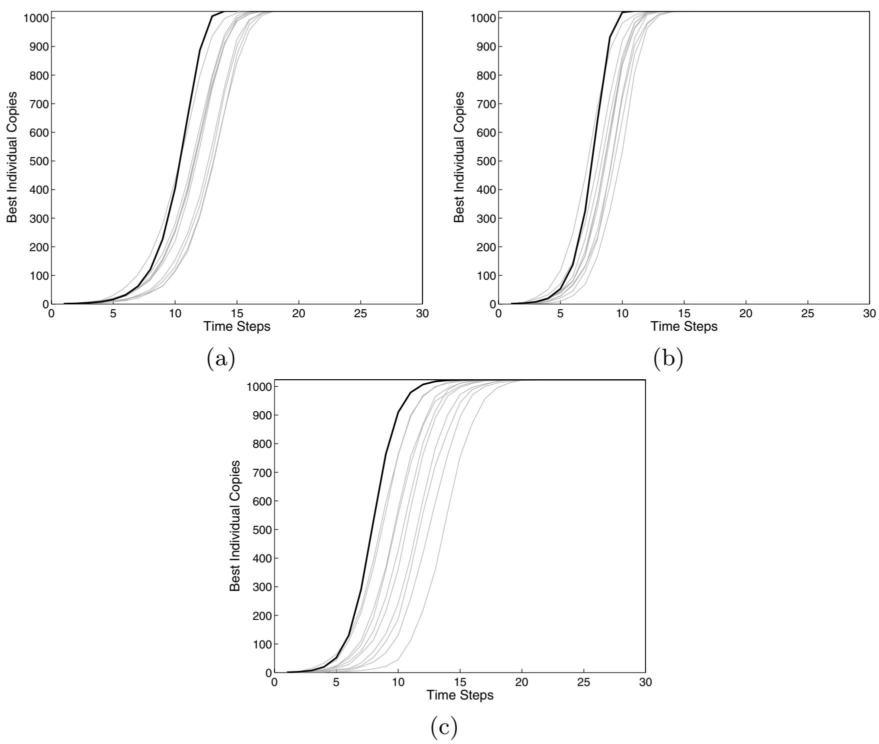


Fig. 6.4. Theoretical curves (black) and experimental curves (gray) for a random-graph population with $q = 0.01$ using synchronous update (a), asynchronous NRS (b), and asynchronous UC (c)

of social games to point out the fact that there is a small separation between any two persons in a social network.

The discovery of these new properties was made possible by the abundance of online network data and the computer power to treat this data, something that was not available to social scientists at earlier times. Now it was finally feasible to measure and analyze "real" networks with hundreds or even tens of thousands of nodes. Following Watts and Strogatz's work, many networks have been studied both man-made and natural: the Internet, the World Wide Web, scientific collaboration and coauthorship networks, metabolic and neural networks, air traffic, telephone calls, and many others. Two recent, excellent reviews of this work are to be found in [9, 111]. Most of these studies have confirmed that, indeed, real networks are not random in the sense of random-graph theory, and they possess a number of extremely interesting properties.

The field of networks is literally exploding, and here I do not have enough space to pursue the big picture any further. The interested reader is referred to the reviews previously mentioned and to the popular books [18, 154] as excellent entry points. Reference [22] contains more advanced material. However, to make the previous considerations more quantitative, to the extent that they can be used for structured populations, we now need to briefly introduce a number of additional concepts related to graphs .

6.4.1 Some Graph Statistics

Drawing and visualizing a network with up to a few tens of nodes may help in understanding its structure. However, when there are hundreds or thousands of nodes this is no longer possible. For this reason, a number of statistics have been proposed to describe the main features of a graph. Taken together, these statistics say a lot about the nature of the network, as we shall see.

Four statistics are particularly useful: the average degree, already defined in Sect. 6.1, the *clustering coefficient*; the *average path length*; and the *degree distribution function*. I shall now briefly describe these graph measures. A fuller treatment can be found in [9, 153].

Clustering Coefficient

There exist two slightly different definitions. The following one is often used. Consider a particular node j in a graph, and let us assume that it has degree k, i.e. it has k edges connecting it to its k neighboring nodes. If all k vertices in the neighborhood were completely connected to each other, forming a clique, then the number of edges would be equal to $\binom{k}{2}$. The clustering coefficient C_j of node j is defined as the ratio between the E edges that actually exist between the k neighbors and the number of possible edges between these nodes:

$$C_j = \frac{E}{\binom{k}{2}} = \frac{2E}{k(k-1)}. \tag{6.7}$$

Thus C_j is a measure of the "cliquishness" of a neighborhood: the higher the value of C_j, the more likely it is that two vertices that are adjacent to a third one are also neighbors of each other. The *average clustering coefficient* $\langle C \rangle$ is the average of C_i over all N vertices $i \in V(G)$: $\langle C \rangle = (1/N)\sum_{i=1}^{N} C_i$. The clustering coefficient of a graph G thus expresses the degree of locality of the connections.

The clustering coefficient of a random graph is simply $\langle k \rangle \simeq p$, where N is the total number of vertices and p is the probability that there is an edge between any two vertices. The clustering coefficient of a complete graph is 1, since each of a node's neighbors are connected to each other by definition.

For a regular one-dimensional lattice, C is given by the following formula [153]:

$$\frac{3(k-2)}{4(k-1)}, \tag{6.8}$$

where $k \geq 2$ is the (constant) number of nodes that are connected to a given node. C is thus independent of N for a regular lattice, and approaches $3/4$ as k increases.

Average Path Length

We denote the shortest path between nodes $i, j \in V(G)$ by l_{ij}. The average, or mean, path length $\langle L \rangle$ of G is defined as

$$\langle L \rangle = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j>i} l_{ij}. \tag{6.9}$$

The calculation of $\langle L \rangle$ for a large graph is compute-intensive. For this reason, approximate sampling techniques, rather than complete enumeration, are used for large N.

The mean path length gives an idea of "how long" it takes to navigate a network. Random graphs and small-world networks share the property that $\langle L \rangle$ scales as $\log N$ and thus most vertices in these networks are connected by a short path. This is not the case in d-dimensional regular lattice graphs, where $\langle L \rangle$ scales as $N^{1/d}$. For instance, in a ring, $\langle L \rangle$ scales linearly with N and is inversely proportional to k, the number of neighbors.

Degree Distribution Function

The degree distribution $P(k)$ of an undirected graph G is a function that gives the probability that a randomly selected vertex has degree k. $P(k)$ can also be seen as the fraction of vertices in the graph that have degree k. Similar definitions also apply for the in-links and out-links of the vertices in a directed graph. For a random graph with connection probability p, the probability $P(k)$ that a random node has degree k is given by

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}. \tag{6.10}$$

This is the number of ways in which k edges can be selected from a certain node out of the $N-1$ possible edges, given that the edges can be chosen independently of each other and have the same probability p. Thus $P(k)$ is a binomial distribution peaked at $P(\langle k \rangle) \simeq Np$, as already found in (6.1). Since this distribution is strongly peaked around the mean, most nodes will have similar degrees, and low- and high-degree nodes, say a few standard deviations away from the mean, have a negligible probability, since the tails fall off very rapidly. Networks having this degree distribution will thus be rather homogeneous.

But most real networks do not show this kind of behavior. In particular, in scale-free graphs, which seem to be common in real life [9], $P(k)$ follows a power-law distribution: $P(k) = c\,k^{-\gamma}$, where c and γ are positive constants. In these networks, while most nodes have a low degree, there is a small but nonnegligible amount of highly connected nodes, and this has a profound influence on the dynamics of processes structured in this way. Other degree distributions are also possible, but need not concern us here.

The degree distribution function, together with the other statistics, are thus a kind of rough "signature" of the type of network and can be helpful in predicting the main aspects of the properties of the network.

In the following sub-sections, I describe the small-world models that were used to run the numerical experiments described later in this chapter.

6.4.2 The Watts–Strogatz Model

Although this model was a real breakthrough in the technical sense when it appeared, today it is clear that it is not a good representation of real networks such as the World Wide Web or the Internet as it retains many features of the Erdös and Rényi random graph. In fact, various scale-free and other types of graphs [10, 111] have been successively proposed as more faithful description of the kinds of big technological, human, and biological networks we observe. In spite of this, the Watts–Strogatz model, because of its simplicity of construction and richness of behavior, still provides an interesting exercise in artificial systems where there is no "natural" constraint on the type of connectivity.

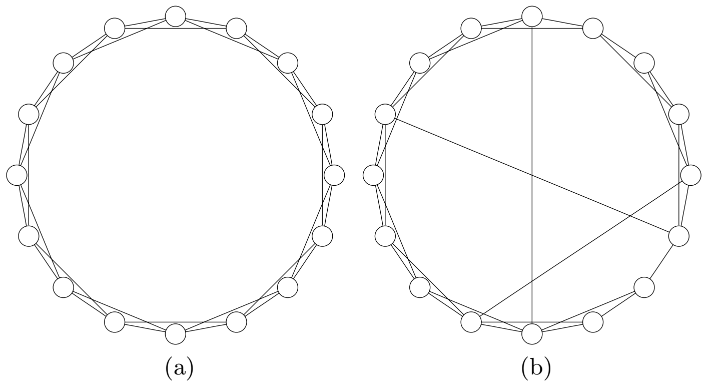


Fig. 6.5. (a) Regular one-dimensional lattice with $k = 4$. (b) A small-world graph obtained by randomly rewiring some of the nearest-neighbor links

According to Watts and Strogatz [153, 155], a small-world graph can be constructed starting from a regular ring of nodes in which each node has k neighbors ($k \ll N$) by simply systematically going through successive nodes and "rewiring" a link with a certain probability β. When the relevant edge is deleted, it is replaced with an edge to a randomly chosen node. If rewiring an edge would lead to a duplicate edge, it is left unchanged. Figure 6.5 schematically depicts this process for a small ring with $k = 4$. This procedure will create a number of *shortcuts* that join distant parts of the lattice. Shortcuts are defined to be edges that join vertices that would be more than two edges apart if they were not connected directly.

Another, nearly equivalent construction starts with a ring lattice with N nodes and k neighbors per node, and shortcut links are added with a certain probability between random pairs of nodes, as depicted in Fig. 6.6.

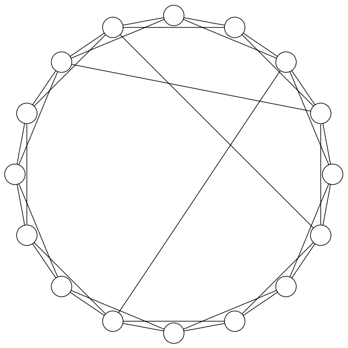


Fig. 6.6. A small-world network obtained by adding a few shortcuts between random vertices in a regular lattice

Shortcuts are the hallmark of small worlds and, while $\langle L \rangle$ scales logarithmically in the number of nodes for a random graph, in Watts–Strogatz graphs it scales approximately linearly for a low rewiring probability β and tends to the random-graph limit as the probability increases[2]. This is due to the progressive appearance of shortcut edges between distant parts of the graph, which obviously contract the path lengths between many vertices. However, small worlds typically have a higher clustering coefficient than do random graphs. Small-world networks have a degree distribution $P(k)$ that is close to binomial for intermediate and large values of the rewiring probability β, while $P(k)$ tends to a delta function for $\beta \to 0$, since in this case we recover the regular lattice.

6.4.3 The Barabási–Albert Model

Albert and Barabási were the first to realize that real networks grow incrementally and that their evolving topology is determined by the way in which

[2] Strictly speaking, the $\beta = 1$ case does not correspond exactly to the Erdös–Renyi random graph. But this is a fine technical point that makes little difference in our arguments (see Chaps. 2 and 3 of [153] for details).

new nodes are added to the network. They proposed an extremely simple model, based on these ideas, that is still useful [9]. One starts with a small clique (a completely connected graph) of m_0 nodes. At each successive time step, a new node is added such that its $m \leq m_0$ edges link it to m nodes already in the graph. When the nodes to which the new node connects are chosen it is assumed that the probability π that a new node will be connected to node i depends on the degree k_i of i such that nodes that already have many links are more likely to be chosen over those that have few. This is called *preferential attachment* and is an effect that can be observed in real networks. The probability π is given by

$$\pi(k_i) = \frac{k_i}{\sum_j k_j},$$

where the sum is over all nodes already in the graph. Barabási and Albert have shown that the model evolves into a stationary scale-free network with a power-law probability distribution of the vertex degree $P(k) \sim k^{-\gamma}$, with $\gamma \sim 3$.

There exist other, more general and more refined models that are capable of producing graphs with a power-law degree distribution (see e.g. [22]). However, the basic Barabási–Albert model is enough for our initial investigation.

6.5 Selection Intensity in Small-World Networks

In all the experiments described below, a population of size of 1024 and a total number of edges of the same order as that of a random graph with $q = 0.01$ were used. The selection mechanism employed was uniform selection in all cases. All the curves presented are averages of 100 independent runs.

6.5.1 Watts–Strogatz Model

For the simulations, Watts–Strogatz small-world networks with 1024 individuals were generated starting from a ring with $k = 10$ neighbors, i.e. a regular radius-5 one-dimensional lattice. In this way the mean number of neighbors is almost equal to that of a random graph with $q = 0.01$.

Figure 6.7 shows growth curves with synchronous update for different values of the rewiring probability β and for the original ring. The trend is clear: when β is increased from 0 (the ring case) to 0.8 (topologies approaching that of a random graph), the selection pressure increases slowly at first, and then very quickly at around $\beta = 0.005$. This can be easily understood if one takes into account how the mean path length and the clustering coefficient vary in a small-world graph. From Fig. 6.8, one can see that for β around 0.005 there is a sudden drop in the average path length from values that pertain to a lattice to values that are close to that of a random graph. This means that, suddenly,

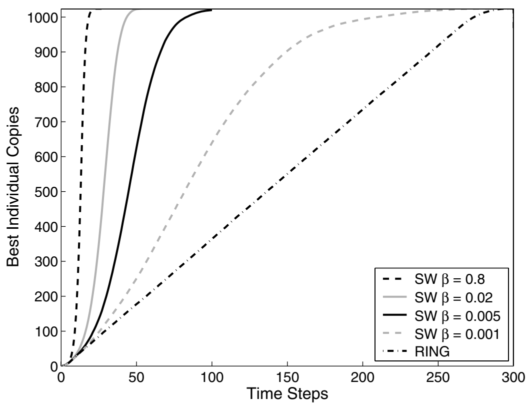


Fig. 6.7. Growth curves for synchronous update with different values of the rewiring probability β. The rightmost curve is for a ring ($\beta = 0$). The leftmost curve corresponds to $\beta = 0.8$, which is almost in the random-graph region

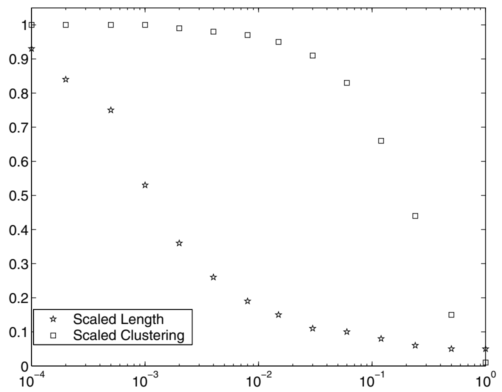


Fig. 6.8. Average path length and clustering coefficient for small-world graphs as a function of the rewiring probability β. Note the logarithmic scale on the abscissa. Redrawn from [153].

many short paths become available through the network between most nodes, which explains the higher growth rate.

Figure 6.9 depicts growth curves for synchronous update, and for the two asynchronous policies in small worlds with $\beta = 0.005$. As in the case of panmictic populations and random graphs, new random sweep is faster than uniform choice, which in turn is faster than synchronous update. The experimental takeover times are to be compared with those of random graphs with the same average number of edges, i.e. Fig. 6.2 d. Clearly, the corresponding small-world graphs induce a lower global selection pressure.

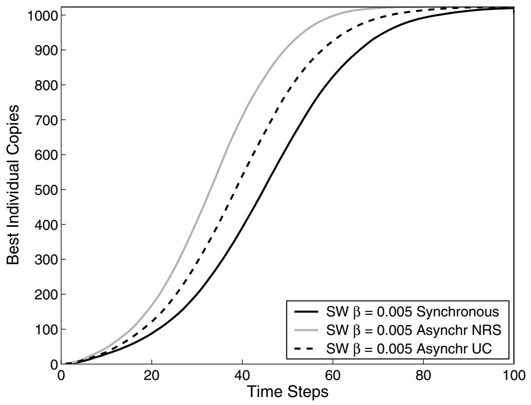


Fig. 6.9. Growth curves for synchronous, new-random-sweep, and uniform-choice asynchronous update. The rewiring probability is $\beta = 0.05$. Note the change in the scale of horizontal axis with respect to Fig. 6.7

6.5.2 Barabási–Albert Model

Scale-free graphs were generated according to the Barabási–Albert model described in Sect. 6.4.3. The starting point was a clique of $m_0 = 14$ nodes, and $1024 - 14 = 1010$ individuals were added, each creating $m = 10$ edges with preferential attachment, following the algorithm.

Figure 6.10 shows the behavior of the growth curves for the three update policies used here. For the position of the initial best individual, any vertex is equally likely. The order of the curves is the same as that observed for random graphs and Watts–Strogatz small-world networks. The inversion in the last part of the uniform-choice curve is due to the fact that cells are chosen with replacement, and thus the last few nonconquered individuals are increasingly unlikely to be chosen. Apart from this effect, the takeover times are very close to those observed for the corresponding random graphs. This confirms that scale-free graphs are a topology in which propagation is at least as fast as in random graphs, which, for example, has important consequences for infection rates [111].

But scale-free graphs have other surprising properties. In particular, those networks are extremely tolerant to attacks on randomly chosen target nodes, which is due to the fact that there are few important (highly connected) nodes and many unimportant (sparsely connected) ones. On the other hand, deliberate suppression of highly connected nodes is likely to produce a lot of damage [9]. The different status of highly connected nodes was demonstrated in an experiment where the initial best individual was always placed on a "hub" node (see Fig. 6.11). In this case the takeover time is very short, shorter than the random-graph case (see Fig. 6.2 and Table 6.1). This is also known to happen in infectious processes, where scale-free communication patterns have the effect of eliminating the so-called infection thereshold [111].

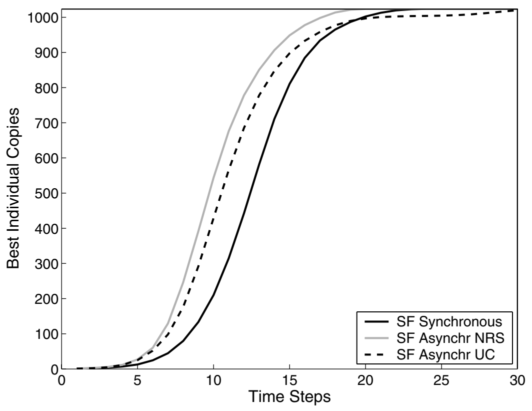


Fig. 6.10. Growth curves for synchronous, new-random-sweep, and uniform-choice asynchronous update in scale-free graphs. The initial best individual was uniformly distributed at random among the nodes

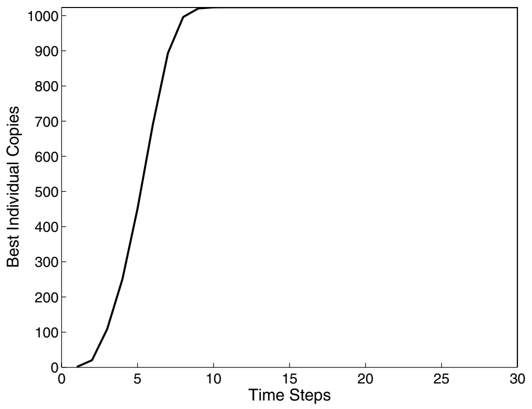


Fig. 6.11. Growth curves for synchronous evolution in scale-free graphs when the initial best individual is placed on a highly connected node

Table 6.2 summarizes the numerical results for the takeover times in Watts–Strogatz and scale-free topologies for the synchronous and the two asynchronous update methods.

It is clear that the effect of small-world topologies on the dynamical properties of processes taking place on a network could be exploited in evolutionary computation by letting the topology adapt or self-organize dynamically in order to control the selection pressure, and thus the explorative or exploitative characteristics of the algorithm.

	Synchronous	Asynchronous NRS	Asynchronous UC
Ring ($\beta = 0$)	279.64 (9.35)	201.51 (9.03)	229.30 (10.54)
WS ($\beta = 0.001$)	168.89 (45.93)	116.50 (27.12)	135.15 (35.10)
WS ($\beta = 0.005$)	80.16 (14.47)	60.18 (8.75)	70.43 (12.08)
WS ($\beta = 0.02$)	45.96 (4.32)	36.58 (4.15)	41.31 (4.27)
WS ($\beta = 0.8$)	19.16 (1.69)	15.43 (1.76)	20.50 (2.46)
BA	17.94 (2.59)	14.60 (2.66)	19.77 (3.55)

Table 6.2. Experimental mean takeover times (with corresponding standard deviations) for the three update methods and for the small-world topologies discussed in the text. WS stands for Watts–Strogatz and BA stands for Barabási–Albert. The experimental results are obtained over 100 independent runs. Population size is 1024 in all cases.

6.6 Summary

In the first part of this chapter, using general stochastic models for the growth of the best individual, we have seen that populations structured as random graphs behave in the same way as panmictic populations of the same size, except when the graph is very sparse. A practical consequence of this result is that it appears unnecessary to use the whole population as a selection pool, given that the random-graph case, using a fraction of the population, shows the same behavior as does the panmictic one. The models and experiments also confirm the results of Chap. 4 for regular lattices, i.e. that the selection pressure can be varied by using different updating schemes.

In the second part of the chapter we examined experimentally the growth of the best individual in two families of networks that are neither regular nor random: the Watts–Strogatz model and the Barabási–Albert model. It appears that the propagation properties of an individual and the global induced selection pressure are qualitatively similar to those found in panmictic populations and random graphs. However, the inhomogeneity of these small-world networks opens up new possibilities for evolutionary computation when the nature of each given node is taken into account. Thus, for example, hubs in scale-free networks largely determine the dynamical properties of the population. By controlling these features, or allowing the population network to self-organize, it is possible to change the selection pressure and thus the characteristics of the algorithm within a wide range. Interesting evolutionary algorithms could be designed along these lines.

7 Coevolutionary Structured Models

7.1 What Is Coevolution?

Straight evolutionary algorithms use the analogy of a single-species population and a suitable definition of a fitness function to rank and select the individuals in the population. The situation in nature is much more complex than what this simple metaphor seems to suggest. Indeed, in biological populations there is a continuous interplay between individuals of the same species, and also encounters and interactions of various kinds with other species. The environment, too, does not stand still, but is continuously changing. This makes the familiar view of a fixed *fitness landscape* (see, for instance, [86] and references therein), in which individuals "climb" through the landscape in order to improve their fitness, hardly adequate. The problem is that the landscape as seen by a given individual is being continually changed by the other individuals and species, and it is not at all clear how a suitable fitness function could be defined.

The points at issue can be clearly seen when one observes such ecological systems as symbiosis, host–parasite systems, and prey–predator systems, in which either organisms mutually support each other, one exploits the other, or they fight against each other. For instance, mutualistic relations between plants and fungi are very common. The fungus invades and lives among the cortex cells of the secondary roots and, in turn, helps the host plant absorb minerals from the soil. Another well-known example is the "association" between the Nile crocodile and the Egyptian plover, a bird that feeds on any leeches attached to the crocodile's gums, thus keeping them clean. This kind of "cleaning symbiosis" is also common in fish.

Other modes of coevolution involve competitive interaction between two specific species. The *Plethodon* salamander is a good example: in the Great Smoky Mountains, two species of salamander compete strongly, as evidenced by the fact that each species will increase its population size if the other is removed. Another classic example is that of foxes feeding on rabbits.

The customary analogy here is the coevolutionary arms race: a plant has chemical defenses, an insect evolves the biochemistry to detoxify these compounds, the plant in turn evolves new defenses that the insect in turn "needs" again to detoxify, and so on. A less innocent example is the coevolution of pathogens in the face of improvements in the effectiveness of antibiotics.

Now, how can we harness the concepts of coevolution for solving problems for which it is difficult or impossible to define a suitable fitness function? A good example is the playing of games. In games such as chess and checkers, there are so many different possible moves that it is practically impossible to exhaustively evaluate all the possible available strategies, given the possible moves and countermoves. One approach is simply to evaluate a player's fitness relative to the other, in a two-player game, as being simply the outcome of the game: won, lost, or perhaps drawn. In this way, a player can iteratively learn, by punishment and reward, to play a better game, without any direct external guidance. This example would be worth pursuing further, but here I would like only to stress the principle: when there is no clear fitness function, a useful strategy is simply to coadapt to the opponent's moves. This situation is similar in spirit to the coevolutionary interactions in nature described above. The interested reader will find an excellent discussion of the issues, and many ideas to harness for coevolutionary problem-solving in Chap. 14 of Michalewicz and Fogel's book [100].

Following an established pattern, I shall divide coevolutionary methods into two main classes: *cooperative coevolution* and *competitive coevolution.* This binary classification is certainly simplistic for biological systems, but it is useful for our purposes in the context of artificial evolution. Cooperative coevolutionary models are artificial evolutionary systems in which a number of different species or groups, each representing a different part of a problem, cooperate in order to find partial solutions, which are then combined in some way to solve the global problem. In competitive coevolution, on the other hand, the different species, populations, or groups prosper or decay at the expense of each other, as typically exemplified by coevolutionary game-playing (see above). Again, the reader is referred to [100] for a fuller discussion.

Most structured coevolutionary algorithms are of the cellular type. Thus, in keeping with the spirit of the main theme of this book, in this chapter we focus on coevolutionary problem-solving using that kind of population topology. But before doing that, a brief description of a cooperative approach is provided next.

7.2 Cooperative Coevolution

There are several examples of artificial cooperative coevolution in the literature. One of the best known is Potter and De Jong's *coadapted-components* architecture [118]. When coadapted components are used, the problem is decomposed into a number of species that each solve a portion, or component, of the global task. The decomposition of the problem is in general not known from the start, and the evolutionary system must be able to address this issue, by using prior knowledge or favoring the emergence of subcomponents. A similar decomposition problem appears in genetic programming, for which techniques that automatically identify blocks of useful code that can be encapsulated and reused have been proposed [122].

A second issue has to do with relationships between components. In the easiest case, components are independent of each other, and can thus be evolved in separate breeding populations. But in most problems of interest there are interdependences among the parts, and these interactions are handled by the architecture by evolving the species in parallel and evaluating them in the context of each other.

In Pott and De Jong's system, each species is evolved in its own population and adapts to the environment through the repeated application of an EA. But the number of species is not fixed: species can disappear when they seem not to contribute enough to the global goal, and new species can be created. Communication between populations (species) is limited to an occasional broadcast of representative individuals.

There are two levels of credit assignment: at the species level, the fitness of an individual is evaluated keeping the representatives of the other species fixed. On the other hand, when an evaluation of the level of the contribution of one species to the global goal is needed, individuals from different species are merged within a shared-domain model to form a composite solution to the target problem.

Finally, in order to be able to apply the model to complex cases in which components need to be represented in different ways, for example in robotics and other fields, heterogeneous representations are supported, such as real parameter encoding, rule systems, and so on.

One interesting feature of the coadapted-components architecture is the natural mapping that there is between the different populations and an island model. Indeed, each species has its EA and communication is sparse, making a multipopulation model, possibly heterogeneous, an obvious one.

This coevolutionary architecture has been applied with success to several problems ranging from function optimization to evolving neural networks, learning sequential decision rules, and more. A recent detailed account of the system's philosophy can be found in [118] and references therein.

7.3 Competitive Coevolution: Hosts and Parasites

The classical work on competitive coevolution is Hillis's pioneering study of sorting networks [76, 77]. Sorting networks are comparison networks that sort their inputs i.e., for any input sequence, the output sequence is monotonically increasing [34]. Thus, if $(a_1, a_2, \dots, a_n)$ is the input sequence and

$(b_1, b_2, \ldots, b_n)$ is the output sequence of the network then we have $b_1 \leq b_2 \ldots \leq b_n$ for every input sequence.

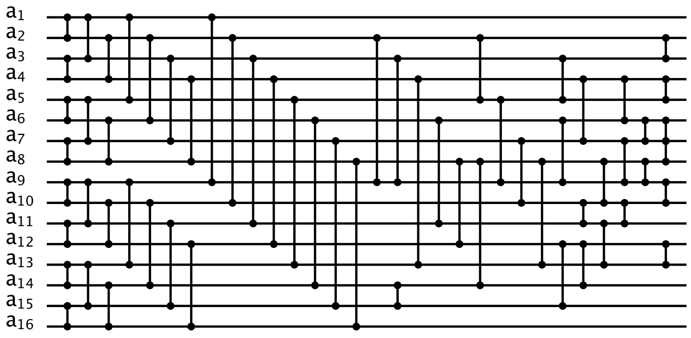


Fig. 7.1. A coevolved network that sorts 16 numbers using 61 comparisons. Redrawn from [77]

The size of the sorting network depends on the number of elements in the list to be sorted. Figure 7.1 depicts graphically one such network for the case $n = 16$, the case that was studied by Hillis. Each horizontal line represents an element of the list to be sorted. Each vertical line represents a comparison to be made between the elements indicated. If the elements are out of order, the network will swap them. Comparisons and exchanges proceed from left to right in the figure, and comparisons in the same column can be made in parallel if truly parallel hardware is used.

A popular problem in the 1960s was to determine the minimum number of comparisons needed for correctly sorting n inputs. The case $n = 16$ was studied in detail, and soon people found networks that did the job with 65 comparisons, then 63 and 62, and, finally, Green found a network that only needed 60, which is the one displayed in Fig. 7.2. This is the best result known to date, but no one has shown yet that it is a lower bound.

As it happens, at the end of the 1980s, Hillis had just finished co-designing and building a massively parallel computer, the Connection Machine CM-2, and commercialized it through a spin-off company. This supercomputer was very famous at the time, for it implemented several innovative ideas in computer architecture. Today, the Connection Machine no longer exists, and this kind of single-instruction-stream, single-data-stream data-parallel computer is restricted to particular computational niches. However, the machine had a distinctive advantage: representing regularly structured networks was very easy with this architecture, by distributing network points over the many simple processors of the machine, one or a few points per processor. The physical machine's communication network, though not as simple as a regular lattice – it was a hypercube – was such that points that were neighbors in the lattice

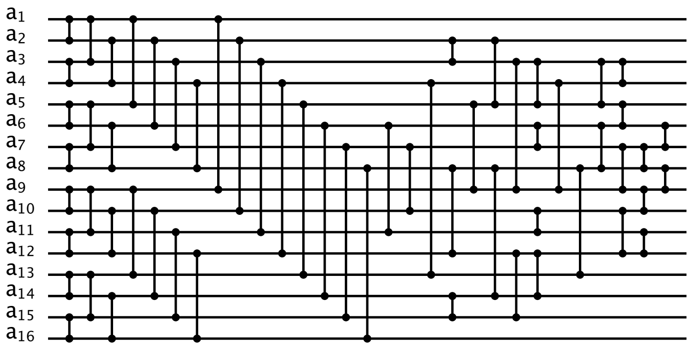


Fig. 7.2. The best sorting network known. It sorts 16 elements using 60 comparisons. Redrawn following [77]

were also stored in neighboring processors at a single vertex of the hypercube. Thus, the machine was eminently suited for homogeneous computational tasks on grids such as cellular automata and discretized partial differential equations. Hillis took advantage of this by using a structured population from the beginning.

In his experiments, Hillis first used a large population of 64 536 sorting networks arranged in a two-dimensional torus topology. The initial population was random except for the fact that part of the chromosome was initialized in such a way as to represent the exchanges that are common to all known sorting networks designed. The encoding of a sorting network is rather involved and will not be reported here. Hillis designed crossover and mutation operators that were tailored to the representation in order to always produce valid networks as offspring. Each individual was ranked according to its percentage of success on the 2^{16} possible binary sequences, since a sorting network that correctly sorts all possible 2^n binary sequences will correctly sort any numerical sequence [34]. Actually, Hillis used a random sample of the possible test cases to attribute a score to a network. Selection and crossover were local. However, instead of using a straight von Neumann or Moore neighborhood, Hillis used short random walks around each individual to select a mate. So, the algorithm was local but the neighborhood was irregular. Using this choice, the best network found had 65 exchanges, a relatively disappointing result. Hillis observed that good solutions spread through the grid and formed patches, separated by crosses formed by bad individuals. This is indeed the usual behavior of cellular EAs.

Two major causes for the relative inefficiency of the process were identified: one was the existence of local optima from which it was difficult to escape, and the second was related to the test cases. After a few generations, most of the test cases were sorted correctly by almost all networks, which means that the fitness differences were small, thus providing little selection pressure on the population. Hillis then decided to make the test cases themselves evolve. In another series of experiments, he added a second population evolving on the same grid as the networks. The second population was made up of individuals which each were a small group of 10 to 20 test cases. Hillis's biological inspiration was that of a host–parasite or prey–predator coevolution. In the artificial case, the parasites were the test cases, and the hosts were the networks. Networks were scored according to how well they sorted the test cases, whereas the parasites were scored for their capabilities to make the networks fail to sort the test cases, i.e. the two fitness functions were complementary.

This coevolutionary algorithm relieves both of the problems found with the straight evolution of networks only. The parallel evolution of the population of parasites makes it more difficult for the system to remain stuck in a local optimum since, as soon as a network becomes rather good at sorting the test cases, it has to make a leap forward, given that the test cases are also improving, and thus making its task harder. The steady evolution of parasites – alias test cases – also means that the test cases that remain in the population are the "hard" ones, those that challenge the sorting ability of the networks better. By harnessing this arms race between networks and test cases Hillis was able to coevolve a network that needed only 61 exchanges, very close to the known best of 60. The resulting network is shown in Fig. 7.1.

Aside from the fact that Hillis's result was valuable in itself, it was above all one of the first that showed convincingly that coevolutionary ideas from biology can be profitably used in problem-solving with artificial evolution. This work thus paved the way for more sophisticated artificial coevolutionary systems, which have since been used in several areas with success (see [100]).

7.4 A Case Study: Coevolution of Cellular Automata

Cellular automata (CAs) are discrete dynamical systems that have been used successfully for simulating physical, chemical, social, and biological systems that are difficult or impossible to model using differential equations or other standard mathematical methods [31]. CAs are constructed from many identical or similar simple components, but their collective behavior may be fairly complex. They are thus an ideal means for investigating basic emergent computational capabilities and other properties of complex systems.

A general problem is to find architectures for a decentralized complex system, such as a CA, that behave in specified ways. However, designing CA rules such that a given macroscopic behavior emerges is not a simple task. Ingenuity and trial-and-error can help, but there is no general methodology that can be applied in all cases.

Evolutionary and coevolutionary algorithms have been quickly recognized as being effective in taking that burden away from the designer by letting the power of artificial evolution automatically find solutions to the problem of the design of CA rules. The work of Packard was one of the first that advocated this idea [112], followed by that of Mitchell and coworkers [103, 102], and Sipper [132, 134]. Here we shall describe only the evolution of CA rules. But both the rules and the connectivity of the network of automata could be made to evolve, giving rise to arbitrary graphs of the kinds described in the previous chapter [136, 147].

7.4.1 Cellular Automata

CAs are dynamical systems in which space and time are discrete. A standard lattice CA consists of an array of cells, each of which can be in one of a finite number of possible states. Here we shall consider only boolean automata, for which the cellular state $s \in \Sigma = \{0, 1\}$. The regular cellular array (lattice) is d-dimensional, where $d = 1, 2, 3$ is used in practice. In a one-dimensional lattice, the topology used here, a cell is connected to r local neighbors (cells) on either side, where r is referred to as the *radius* (thus, each cell has $2r + 1$ neighbors, including itself).

CAs are usually updated synchronously in discrete time steps, according to a local rule, identical for all cells. The state of a cell at the next time step is determined by the current states of the surrounding neighborhood of cells, including the cell itself:

$$s_i^{t+1} = f(s_{i-r}^t, \ldots, s_i^t, \ldots, s_{i+r}^t), \qquad f : \Sigma^{2r+1} \to \Sigma,$$

where s_i^t denotes the value of site i at time t, $f(.)$ represents the local transition rule, and r is the radius of the CA. The term *configuration* refers to an assignment of ones and zeros to all the cells at a given time step. It can be described by $\mathbf{s}^t = (s_0^t, s_1^t, \ldots, s_{N-1}^t)$, where N is the lattice size. Often CAs have periodic boundary conditions $s_{N+i}^t = s_i^t$.

Configurations evolve in time according to a global update rule Φ, which applies in parallel to all the cells: $\mathbf{s}^{t+1} = \Phi(\mathbf{s}^t)$.

This is the model investigated in this chapter, together with a simple extension termed *nonuniform cellular automata* [134]. Such automata function in the same way as uniform ones, the only difference being that the cellular rules that need not be identical for all cells.

7.4.2 The Majority Task

Evolutionary algorithms have been used to help evolve suitable rules for several CA tasks. Here we concentrate on a theoretically important example, the *density task*, also called the *majority task*. The majority task is a prototypical distributed computational task for CAs. For a finite CA of size N, it is defined as follows. Let ρ^0 be the fraction of ones in the initial configuration (IC) $\mathbf{s}^0$. The task is to determine whether ρ^0 is greater than or less than $1/2$. If $\rho^0 > 1/2$ then the CA must relax to a fixed-point configuration of all ones;

otherwise it must relax to a fixed-point configuration of all zeros, after a number of time steps of the order of the lattice size N (N is odd to avoid the case $\rho^0 = 0.5$, for which the result is undefined). This computation is trivial for a computer having a central control. Indeed, just scanning the array and adding up the number of, say, bits equal to 1 will provide the answer in $O(N)$ time. However, this is nontrivial for a small-radius one-dimensional CA, since such a CA can only transfer information at finite speed, relying on local information exclusively, while the density is a global property of the configuration of states [103]. Figure 7.3 shows the operation of a CA obtained through artificial evolution.

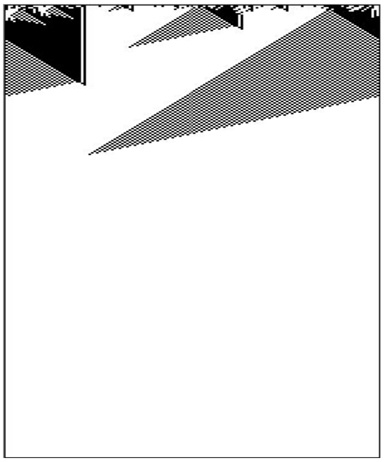


Fig. 7.3. The operation of an evolved one-dimensional, radius-3 CA for the density task. The CA cell states are represented horizontally (black stands for 1 and white for 0). Time increases down the page. The CA rule was obtained through artificial evolution by Mitchell et al. [102]. The density ρ^0 is 0.416 and the lattice size N is 149

It has been shown that the density task cannot be solved perfectly by a uniform, two-state CA with a finite radius [91], although a slightly modified version of the task can be shown to admit a perfect solution by such an automaton [28] or a combination of automata [59].

The *performance* P of a given CA on the majority task is defined as the fraction of correct classifications over 10^4 randomly chosen ICs. The ICs are sampled according to a binomial distribution (i.e. each bit is independently drawn with a probability 1/2 of being 0). Clearly, this distribution is strongly peaked around $\rho^0 = 1/2$ and thus it creates a difficult case for the CA to solve.

The lack of a perfect solution does not prevent one from searching for imperfect solutions of as good a quality as possible, given also that no upper bound on classification accuracy has been found. In general, given a desired global behavior for a CA – for example, capability to solve the density task – it is extremely difficult to infer the local CA rule that will give rise to the emergence of a desired computation, owing to possible nonlinearities and large-scale collective effects that cannot in general be predicted from the local CA updating rule alone. Since exhaustive evaluation of all possible rules is out of the question except for elementary ($d = 1, r = 1$) automata, one possible solution to the problem is to use evolutionary algorithms, as first proposed by Mitchell et al. [102, 103] for uniform CAs and by Sipper for nonuniform ones [133].

The density task is actually a form of machine-learning problem, since the CA must approximately solve the problem on the basis of only a relatively small sample of ICs. given that for $N = 149$, the value used here, there are 2^{149} possible ICs. There is thus a problem of generalizing the results to unseen ICs, and there is also the question of scalability of the task: how does a given evolved CA rule, or combination of rules in the nonuniform case, perform as N becomes larger and larger?

7.5 Artificial Evolution of CAs for the Majority Task

The early work of Mitchell et al., employing a standard GA, showed that evolution of CAs for the density task is possible but it is hard (see [35] for a recent review). Indeed, only a small fraction of the evolutionary runs gave rise to good-quality automata. This is an indication either that the problem space is difficult to search, or that a standard GA is not entirely suitable for the search, or both. Actually, recent research has shown that the space of automata is indeed a hard one to search, and thus any heuristic will face difficulties [151].

Mitchell and Crutchfield's group has been able to identify three main types of CA strategy that a GA tends to evolve. The first type is called "default strategies"; these classify all bit strings into a single density class, either the class of all zeros or all ones. Of course, these rules have a performance of around 0.5 since about half of the random IC sample will be classified correctly. They are thus no better than simply classifying an IC by throwing an unbiased coin. The second type is made up of CAs that employ "block-expanding" strategies. Block-expanding strategies are a little better than default strategies, as they rely on the presence on sufficiently large blocks of zeros or ones in the IC. If such blocks exist, these strategies tend to converge to the corresponding fixed point. Since the existence of sizable blocks of zeros or ones is correlated with the density of the IC, there is more than a 0.5 chance that the CA will converge to the right fixed point. Block-expanding strategies rely mainly on local information and are not robust in global terms. Both default and block-expanding strategies are easily evolved, but they are unsatisfactory.

In a small number of cases, a third type of much more sophisticated strategies have been identified that are able to achieve the desired global coordination required for approximately solving the density task with high efficacy: the "embedded-particle" strategies. These strategies rely on the existence of spatio-temporal propagating patterns that can be made visible after filtering the "uninteresting" regular background. Crutchfield and coworkers have shown how global CA computations can be interpreted in terms of those propagating particles and their interactions for some global CA tasks, including the majority task. (see [35] and references therein for a detailed explanation).

In spite of the difficulty of the problem, evolutionary algorithms have been relatively successful on the density task, and several groups have been able to find CAs with a performance of around 0.8 or more. This work is discussed in detail in [35] and will not be reported here. Some of the best CAs were obtained with coevolutionary algorithms. For example, Juillé and Pollack, using a coevolutionary approach, obtained $P = 0.86$ for the standard ring size $N = 149$ which, to the best of the author's knowledge, is the best result at the time of writing [85].

Despite the success of coevolutionary algorithms, it has been observed that sometimes they are unable to provide continuous progress in the performance of the populations, giving rise to stagnation and lockup into average performance states or to fluctuations among relatively mediocre solutions. According to [85], this may be due to an inability of the changing environment to provide useful information for the learners to be able to further improve their behavior. One of the main reasons for this state of affairs is the phenomenon known by the colorful name of "Red Queen effect" [115]. Roughly speaking, the effect consists of the following: individuals are evaluated in a constantly changing environment of fitness cases; as a result, they tend to perform well against the current population of test cases, forgetting some of the traits that made them effective on other groups of test cases seen before. Thus, the two populations perform well against each other but they are less effective against opponents chosen from outside. In other words, they have poor generalization capabilities. It is as if the individuals did not care about solving the original optimization problem, and were busy just finding ways to beat current good representatives of the other population. Several remedies have been suggested to improve the effectiveness of straight competitive coevolution. Here, in keeping with our emphasis on structured EAs, I shall describe coevolutionary approaches that make use of spatially structured coevolving populations.

7.5.1 Coevolving Uniform CAs for the Majority Task

In [113], Pagie and Hogeweg compared spatially structured coevolving populations with completely mixing ones for the purpose of evolving good solutions to the majority-task problem. In both cases they used two antagonistic populations, one representing two-state, radius-3 CAs, while the other was formed from the initial configurations that the CAs worked with. This is entirely analogous in spirit to Hillis's host–parasite approach for sorting networks described above, with the CAs playing the role of hosts and the ICs that of parasites.

As in the original work of the Santa Fe group, in Pagie and Hogeweg's study both CAs and ICs are represented as bit strings. A CA rule is encoded as a bit string containing the next-state (output) bits for all possible neighborhood configurations, listed in lexicographic order; for example, for CAs with $r = 1$, the genome consists of eight bits, where the bit at position 0 is the state to which neighborhood configuration 000 is mapped, and so on until bit 7, corresponding to neighborhood configuration 111. The rule number is the number that is obtained when the string of the rule's output bits is interpreted as a decimal number. In the present case, with a radius-3 CA (seven neighbors), the rule table is of size $2^7 = 128$.

The topology of the two populations is a two-dimensional lattice of size 30×30 with periodic boundary conditions, with each lattice point representing one CA and one IC. The population of automata starts with random CAs represented by bit strings generated according to a binomial distribution, while the ICs are initialized with all-zero bit strings. Evolution is done synchronously. In the mixing model, the topology of the two populations is the same but the individuals are randomly shuffled in the grid at each time step, which approximates a panmictic situation.

The fitness evaluation and genetic operators are local. The fitness of a CA is based on the nine ICs in its Moore neighborhood. CAs are scored by running them on the nine neighbor ICs for about $2N$ steps each. Credit is given only in the case of successful classification. With respect to the standard GA approach, where each CA is evaluated on hundreds of ICs, here the fitness evaluation is sparse. This does not seem to be a drawback, however. In fact, sparse fitness evaluation for both the CAs and the ICs seems to be a positive factor for the evolutionary process, as was the case in Hillis's work. The fitness of an IC is based on the CA located in the same cell only.

Previous studies using coevolution for the majority task showed a tendency for the ICs to quickly evolve toward densities around 0.5. This is understandable, since the ICs are trying to make the task of the classifier CAs harder and harder. To avoid this problem, Pagie and Hogeweg used a density-dependent fitness function that gives more weight to low and high density values.

Selection is probabilistic in Pagie and Hogeweg's study, based on the rank of the nine individuals in the Moore neighborhood of a given individual. Each CA in the population is replaced by the winner of a tournament including that CA itself and its eight neighbors, using the rank order of the individuals. ICs are selected in the same way. After selection and replacement, bit-flip mutation is applied to both CAs and ICs with a rate of 0.0016 per bit for CAs and 0.0034 per bit for ICs. These values correspond to an expected number of 0.2 mutation events for CAs and 0.5 for ICs, when multiplied by the population size (900). Crossover is not used.

The results of this empirical investigation were rather interesting. The mixing model was never able to evolve CAs that employed sophisticated classification strategies, and the best performance values were always around 0.5–0.6 and were obtained with default or block-expanding strategies. The performance values of the best CAs coevolved in the spatially local model were much better, around 0.75, and corresponded to CAs that used more complex particle-based strategies to solve the problem.

The evolutionary dynamics observed when individuals remained localized in space and when they were globally mixed at each time step were extremely different. Red Queen dynamics were seen to occur in the mixing model, characterized by oscillations of the types of CAs and ICs that were present in the population, and an ensuing loss of efficiency in the search. On the other hand, in the spatially embedded coevolutionary model, spatial patterns could form with well-defined frontiers that changed slowly for both the CA and the IC population. In this case, the interactions between individuals remain local in space and time, giving rise to many competitions between local species in the patches. This enhances the efficiency of the evolutionary process in the sense of the optimization and generalization capabilities of the resulting CAs. Diversity measures in the two models support these findings. Once again, it has been empirically confirmed that the spatial structure of populations may promote diversity and speciation for “free”, so to speak, without the use of complicated explicit sharing and modified-fitness techniques.

Pagie and Hogeweg’s investigation shed some light on the mechanisms at work in a spatial coevolutionary algorithm with two antagonistic species, but was not intended to find highly optimized classifiers for the majority-task problem. In a follow-up study, Pagie and Mitchell [114] compared the spatially embedded coevolutionary model described above with a straight evolutionary spatial model of the same kind. The size of the grids was the same, 30×30, but in the second model only the CAs evolved. The other EA parameters, including the fitness evaluation, were the same as in the previous study, except that, in the evolutionary setting, the population of ICs did not evolve but was generated anew at each time step, being selected at random from a uniform density distribution in which each density in $[0, 1]$ was equally likely. By the way, this is how ICs were produced at each generation in the original panmictic GA [103] .

Pagie and Mitchell found that the coevolutionary spatially structured model led to a much higher frequency of evolution of CAs employing sophisticated strategies than did the standard structured evolutionary algorithm. In fact, out of a total of 100 runs, coevolution found particle strategies 86 times, while straight evolution was successful only two times. Of course, both models evolved default and block-expanding strategies in all runs. These results are striking: the evolutionary model is nearly always unable to make the transition to effective density classification strategies, while the coevolutionary model does so most of the time.

Pagie and Mitchell attributed the difference between the two models to a mechanism related to the way in which ICs are generated. In the evolutionary model, block-expanding CAs can prosper in the population because they are evaluated on ICs that are randomly generated according to a uniform distribution of densities. However, these CAs generalize poorly when run on ICs drawn from a binomial distribution. In the coevolutionary model, on the other hand, block-expanding CAs can be fooled and exploited by the population of coevolving ICs. This occurs because an IC with low density of zeros or ones can incorporate a "deceptive" block of bits of the same type, which would lead block-expanding strategies to reach the wrong fixed point and thus to misclassify the IC. This effect lowers the fitness of the current generation of block-expanding CAs and pushes the system to discover more sophisticated strategies that do not fall into the trap of deceptive blocks. More generally, Pagie and Mitchell remark that in the coevolutionary model, ICs evolve in such a way as to exploit the population of CA rules in the case of particle strategies also. In order to evaluate this effect, in another set of runs they modified the coevolutionary model such that no particular bit patterns could form. This was done by representing ICs as density values, rather than explicit bit strings. The modified coevolutionary model was less efficient than the original one: particle strategies were evolved in 23 out of 100 runs instead of 86% of the time. However, it was still more efficient than the plain evolutionary model.

The higher efficiency of the original coevolutionary model has a downside in that the out-of-sample performance P of coevolved CAs is somewhat lower than for CAs coevolved with the modified model. Indeed, the average P value was 0.76 for the former and 0.80 for the latter. The maximum values reached about 0.85. The explanation is that in the original model, ICs evolve particular bit patterns which exploit weaknesses in the CA population. As a result, the CAs also undergo selection to adapt to these ICs. But these strings, being atypical, are not present in large numbers in the out-of-sample ICs, on which performance is ultimately measured.

7.5.2 Coevolving Nonuniform CAs for the Majority Task

Sipper [132, 133] introduced a coevolutionary approach for evolving nonuniform CAs for several computational tasks, including the density task. Nonuniform cellular automata relax the constraint that each cell contains the same rule, allowing different rules for different cells. The use of nonuniform CAs opens up new possibilities for global cellular computing, increasing the degrees of freedom that evolution can play with. In fact, for a one-dimensional CA of size $N = 149$ and radius 3, the search space has a size 2^{128} for uniform CAs, while the size increases to $(2^{128})^{149}$ in the case of a nonuniform grid, a huge search space, vastly larger than the search space of uniform CAs. It is a remarkable fact that, in spite of this more than astronomical size, structured coevolution manages to find high-performance solutions for the density task and for other problems. This may be due in part to the fact that the increased freedom in choosing cell rules makes many different nonuniform CAs more or less equally good at the task, and we shall see that this added freedom in cell rules will permit a reduction of the CA radius. The coevolutionary approach suggested by Sipper is called the *cellular programming algorithm* and is explained below.

The Cellular Programming Algorithm

A cell's rule table is encoded as a bit string in Wolfram's notation, as explained at the beginning of Sect. 7.5.1. Rather than employing a *population* of evolving, uniform CAs, as with genetic algorithm approaches, cellular programming involves a *single*, nonuniform CA of size N, with cell rules initialized at random. Initial configurations are then generated at random, in accordance with the task at hand, and for each one the CA is run for $M = O(N)$ time steps. Each cell's *fitness* is accumulated over C initial configurations, where a single run's score is 1 if the cell is in the correct state after M iterations, and 0 otherwise. After every C configurations, evolution of rules occurs by application of crossover and mutation. This evolutionary process is performed in a completely *local* manner, where genetic operators are applied only between directly connected cells. It is driven by $nf_i(c)$, the number of fitter neighbors of cell i after c configurations. The pseudocode of the algorithm is shown in Fig. 7.4.

Crossover between two rules is performed by selecting at random (with uniform probability) a single crossover point and creating a new rule by combining the first rule's bit string before the crossover point with the second rule's bit string from this point onward. Mutation is applied to the bit string of a rule with a probability of 0.001 per bit.

There are two main differences between the cellular programming algorithm and the standard genetic algorithm. (a) The latter involves a population of evolving, uniform CAs; all CAs are *ranked* according to fitness, with crossover occurring between *any* two individuals in the population. Thus, while the CA runs in accordance with a local rule, evolution proceeds in a *global* manner. In contrast, the cellular programming algorithm proceeds *locally* in the sense that each cell has access only to its locale, not only during the run but also during the evolutionary phase, and no global fitness ranking is performed. (b) The standard genetic algorithm involves a population of *independent* problem solutions; the CAs in the population are assigned fitness values independent of one another, and interact only through the genetic operators in order to produce the next generation. In contrast, in cellular programming the CAs in the grid *coevolve* since each cell's fitness depends upon its evolving neighbors. This may also be considered a form of symbiotic cooperation. Actually, the approach can be considered both as a cooperative coevolutionary approach and, at the same time, a competitive one in the sense that rules are also competing for "room" in the lattice.

```
for each cell $i$ in CA do in parallel
    initialize rule table of cell $i$
    $f_i = 0$ { fitness value }
end parallel for
$c = 0$ { initial configurations counter }
while not done do
    generate a random initial configuration
    run CA on initial configuration for $M$ time steps
    for each cell $i$ do in parallel
        if cell $i$ is in the correct final state then
            $f_i = f_i + 1$
        end if
    end parallel for
    $c = c + 1$
    if $c$ mod $C = 0$ then { evolve every $C$ configurations}
        for each cell $i$ do in parallel
            compute $nf_i(c)$ { number of fitter neighbors }
            if $nf_i(c) = 0$ then rule $i$ is left unchanged
            else if $nf_i(c) = 1$ then replace rule $i$ with the fitter neighboring rule,
                                   followed by mutation
            else if $nf_i(c) = 2$ then replace rule $i$ with the crossover of the two fitter
                                   neighboring rules, followed by mutation
            else if $nf_i(c) > 2$ then replace rule $i$ with the crossover of two randomly
                                   chosen fitter neighboring rules, followed by mutation
                                   (this case can occur if the cellular neighborhood includes
                                   more than two cells)
            end if
            $f_i = 0$
        end parallel for
    end if
end while
```

Fig. 7.4. Pseudocode of the cellular programming algorithm

Note also that there is a difference between the cellular programming algorithm and classical cellular EAs or coevolving EAs such as those described in Sect. 7.5.1. The latter are also structured and local, but the population represents different potential solutions to the problem, whereas the CA grid is "the solution" when the cellular programming algorithm terminates.

7.5.3 Results for the Density Task

Recall that Mitchell et al., as well as the other researchers, found that a lattice radius of 3 was needed for a uniform CA to approximately solve the majority task with high performance. Sipper studied the task using nonuniform, one-dimensional, minimal-radius ($r = 1$) CAs of size $N = 149$. Owing to the nonuniformity, the search space involved is still vastly larger than the 2^{128} of radius-3 uniform CAs. In fact, since each cell contains one of 2^8 possible rules this space is of size $(2^8)^{149} = 2^{1192}$. The most important result is that evolved, nonuniform, $r = 1$ CAs show high performance on the task, rivalling that of uniform radius-3 CAs, in spite of their smaller radius (see [132, 133] for details).

The cellular programming algorithm used randomly generated initial configurations, uniformly distributed over densities in the range $[0, 1]$, with the CA being run for $M = 150$ time steps (thus, computation time is linear with grid size). It was found that the nonuniform CAs that coevolved consisted of a grid in which one rule dominated, a situation referred to as quasi-uniformity [133]. Basically, in a *quasi*-uniform CA, the number of distinct rules is extremely small with respect to the rule-space size; furthermore, the rules are distributed such that a subset of dominant rules occupies most of the grid.

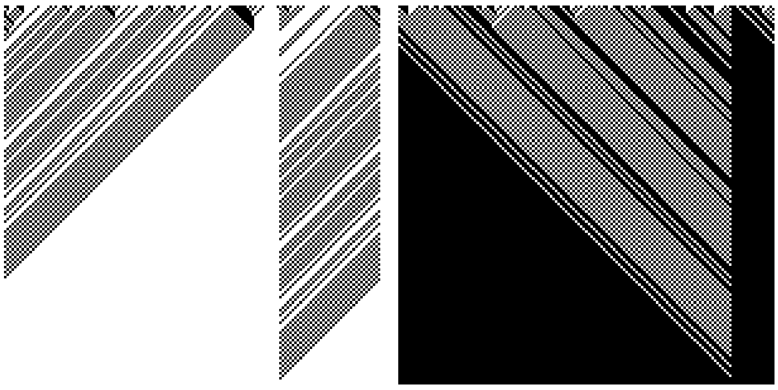


Fig. 7.5. One-dimensional density task: operation of a coevolved nonuniform $r = 1$ CA. The grid size is $N = 149$. White squares represent cells in state 0, and black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). Initial configurations were generated at random. (a) Initial density of ones is 0.40. (b) Initial density of ones is 0.60. The CA relaxes in both cases to a fixed pattern of all zeross or all ones, correctly classifying the initial configuration. Reproduced from [135] with the author's permission

Figure 7.5 demonstrates the operation of one such coevolved CA. In this example the grid consists of 146 cells containing rule 226, two cells containing rule 224, and one cell containing rule 234. The nondominant rules act as "buffers", preventing information from flowing too freely, and making local corrections to passing signals.

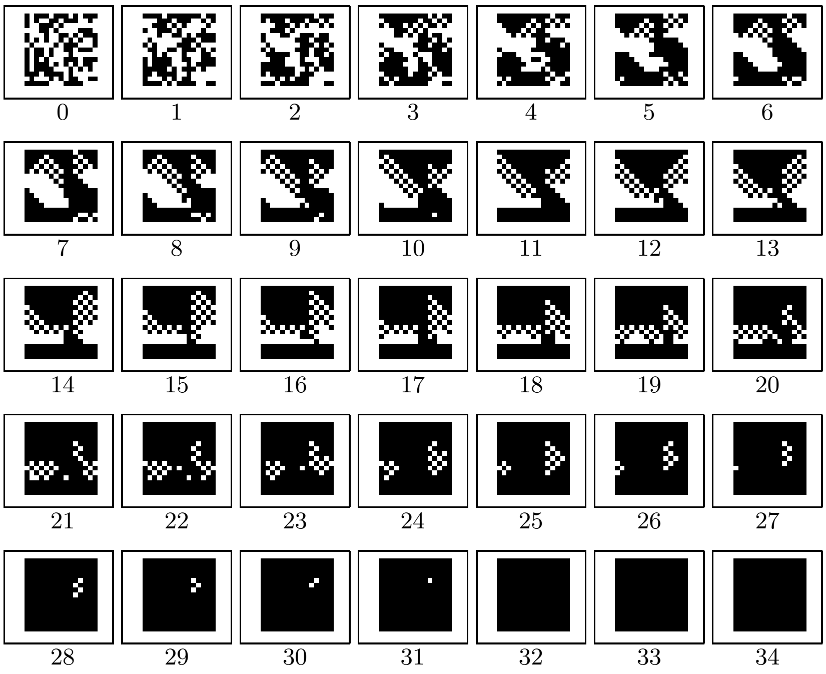


Fig. 7.6. Two-dimensional density task: operation of a coevolved nonuniform two-state, five-neighbor CA. The grid size is $N = 225$ (15×15). The initial density of ones is 0.51, and the final density is 1. The numbers at the bottom of images denote time steps. Reproduced from [135] with the author's permission

The density task can be extended in a straightforward manner to two-dimensional grids, an investigation that Sipper carried out [133], attaining notably higher performance than in the one-dimensional case. Furthermore, computation time, i.e. the number of time steps taken by the CA until convergence to the correct final pattern, was shorter. Figure 7.6 demonstrates the operation of one such coevolved CA. Qualitatively, we observe the CA's "strategy" of successively classifying local densities, with the locality range increasing over time; "competing" regions of density 0 and density 1 are manifest, ultimately relaxing to the correct fixed point.

7.6 Summary

The conclusion of this chapter is that coevolutionary algorithms are an extremely rich and powerful heuristic for machine learning and coadaptation. Coevolution can indeed significantly improve the results of an evolutionary search and is probably the only practicable approach when there is no knowledge about a suitable fitness function. Coevolution also has a few drawbacks, related to the Red Queen effect and the persistence of mediocre stable states. However, ways have been found to avoid most pitfalls.

Another empirical observation is that, once again, structured coevolving populations seem to offer implicit advantages with respect to panmictic ones. Confirming the observations in previous chapters, the benefits seem to be related to the slow mixing and the preservation of diversity offered by the lattice structures. For example, we have seen how the debilitating Red Queen effect that is so often present in coevolutionary dynamics is mitigated by the use of structured populations.

Finally, the case study of CA rule evolution for the majority task showed that coevolution of structured populations offers several distinct advantages for this difficult problem, in the form of both competitive coevolution and the cooperative scheme of the cellular programming algorithm.

8 Some Nonconventional Models

In this last chapter I would like to go beyond the established, relatively straightforward structured-population models and provide a glimpse of other, more complex models that have been proposed. Indeed, nature is by no means mean in providing ideas and inspiration, and evolutionary-computing researchers are very imaginative. As a result, many new models have appeared, some of which seem promising. As mentioned before, we prefer simpler models, when they work well, because those are the ones that are easier to analyze mathematically. The price to pay for using more complex ideas and models is just that: even when they work satisfactorily, they are more difficult to understand and to describe. However, the fact that we are unable to really understand the dynamical properties of these models today should not prevent us from using them, as long as they prove efficient. Some day we might be better equipped to analyze their behavior and we could then include some of those models in our problem-solving toolkit. After all, 20 years ago even simple EAs were poorly understood by today's standards, and we are now in the position of using them nearly routinely.

As a matter of terminology, the coevolutionary structured models of the previous chapter might also be considered nonstandard. After all, the subpopulations or network vertices are not homogeneous, as was the case for all the elementary models described earlier. However, given the large amount of research that exists and their particular features, coevolutionary structured EAs well deserve a chapter of their own. The same is true for the general graph population topologies described in Chap. 6. In addition, topology-modified homogeneous structured models using more complex, perhaps hierarchical network topologies for communication can also be considered nonconventional. However, as explained in Sect. 1.2.3, we still know too little about these models, and therefore I prefer to leave them aside.

8.1 Nonconventional Island Models

A number of nonconventional models based on sparsely communicating subpopulations have been proposed. They operate at the level of either the evolutionary parameters, the population composition, or the hierarchy of exchanges between populations. For the sake of illustration, I have chosen to present two models out of the many that have been proposed: the injection island model and a system that uses variable-size populations.

8.1.1 The Injection Island Model

The injection island GA was proposed by Lin et al. in [93]. It is an extension of the customary island model GA in which the subpopulations are not homogeneous; rather they search at different levels of resolution in the problem space. Also, the subpopulation fitness function amd the representation of individuals may differ among islands. The idea is to search at a lower resolution in some of the islands and "inject" high-performance individuals into higher-resolution islands to fine-tune the solutions. In general, low-resolution evaluation will be less demanding and costly than fitness evaluation at higher levels. At the same time, different GA parameters may be used in different-level islands, including mutation and crossover rates, as well as communication rates.

The different resolutions of the GA representation are obtained by encoding problem solutions with different numbers of bits. Thus an injection island GA has multiple subpopulations that encode the same problem using different block sizes.

Each subpopulation searches independently for a good individual at its own resolution. At regular intervals, individuals are exchanged between populations. Exchanges are only allowed up the ladder from a lower-resolution to a higher-resolution node. This is reasonable, since in going from a low-resolution to a high-resolution island translation to the appropriate block size is required, and this can be done without loss of information only from lower- to higher-resolution. On the basis of this rule, Lin et al. considered several static communication topologies for sending individuals. An example is depicted graphically in Fig. 8.1. In the figure, islands with different levels of resolution evaluate fitness using increasingly precise and sophisticated definitions, which are correspondingly increasingly computationally expensive.

Lin et al. claim several advantages for the injection island GA with respect to both standard panmictic GAs and homogeneous island GAs. They enumerate the following beneficial effects:

- Building blocks of lower resolution can be found and maintained at that resolution level. They can later be refined by a child island at a higher resolution.
- The search space size at lower levels is effectively reduced in size, which allows one to find promising solutions faster, and then inject them into higher resolution islands for refinement.

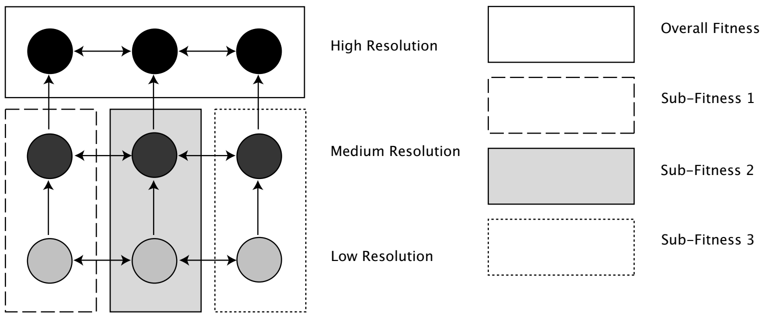


Fig. 8.1. Schematic representation of an injection island GA. The islands at each level can communicate among themselves and search at that level of resolution. Vertical arrows show communication from lower- to higher-resolution islands. Accuracy increases up the ladder, while computational efficiency is better at low resolution Redrawn from [44].

- Nodes that are connected in the ladder share portions of the same search space and can thus cooperate in finding and refining good solutions.
- The algorithm favors finding an efficient problem partitioning, through a divide-and-conquer strategy.

The injection island approach has been tested on difficult benchmark problems, including deceptive functions, and on real-life problems [44]. An application to a difficult engineering problem, in particular, showed that the methodology of searching at different levels of resolution is robust and outperforms standard island GAs in terms of solution quality and computer time. The structure of the algorithm can also easily accommodate searching for a multicriterion objective function by using each individual criterion as the fitness function of a subpopulation. On the other hand, compared with a standard island GA, the structure of the algorithm requires more care in choosing appropriate representation levels and in tuning the model's parameters.

8.1.2 Islands with Variable Population Size

Evolutionary algorithms, owing to their population structure, expend a comparatively large amount of computational effort to solve problems. This "population" effect is more marked when algorithms use variable-size chromosomes, which has been traditionally the case in genetic programming. In fact, it has been observed that GP individuals tend to steadily grow in size as generations are computed. The phenomenon goes under the name of *bloat*, and there have been several proposals aimed at preventing such an inordinate growth (see [92] for a rather complete survey). For instance, the fitness function may embody a penalty associated with the size of the individuals, so that short individuals are favored. Another proposal is to set a limit on the maximum size. Multiobjective techniques have also be applied to GP in such a way that both the size of individuals and the fitness are considered as simultaneous criteria to be optimized [38]. Most of these techniques have some drawbacks: either they change the way in which fitness is computed, thus influencing the structure of the fitness landscape and the main characteristics of the problem, or they require additional computational costs to be implemented, often thwarting their positive effects on size. Moreover, most proposed solutions focus on the size of individuals and do not take into account the global problem of population growth as a whole.

In some studies [52, 53, 120], bloat and the reduction of computational effort have been approached employing another perspective: given that the increase in the size of individuals produces an overall growth in the population, the population size is made variable, in an attempt to control the computational effort required for evaluating individuals. Variable-size populations have seldom been used: two studies pertaining to GAs are [13, 45]. Another structured model, called the *patchwork* model [90], will be described in Sect. 8.2.1.

Adding and Suppressing Individuals

If one looks at a typical EA run, one usually sees that fitness improves quickly at the beginning of the run, while it tends to level off later in the run. This suggests that many individuals of low quality do not contribute to the overall population fitness at the beginning, while individuals tend to become fitter but more uniform and do not provide sufficient evolvability later in the run. In practice, since the population is partially converged, it is as if there were *fewer* individuals. This leads to the idea of deleting individuals when fitness is still improving vigourously, and of adding new individuals when the population stagnates. Of course, one would like the system to do this automatically i.e. that it should self-adapt its population size during the run.

In [120] it was suggested that the number of individuals to be deleted or added should be some function of the slope of the fitness versus computational effort curve. When fitness is rapidly improving, a proportionally larger number of individuals is deleted. When fitness tends to stagnate and the slope of the curve is less than a given value, individuals are added to the population. Of course, the numerical details can vary widely. The original reference contains details about the actual numerical expressions used and how they were chosen.

An important question is: which individuals are deleted and where do new individuals come from? To avoid losing good genetic material and to fight bloat, given a number of individuals to be deleted, they are chosen from the worst in the population, and of those, the largest are selected. Individuals to be added are not generated anew; instead, they come from another island and are chosen from the best in that island. Their number is determined in such a way that the original population size times a constant is not exceeded. Again, fine details can be found in the original reference.

Results for the Test Problems

The test problems described in Chap. 3 were used in the studies described in what follows: the symbolic regression problem, the artificial ant problem and the even-parity problem. The parameters of the GP system were also the same as before. The topology of the system comprised five islands connected according to a random communication topology. The only difference from the standard model was the presence of a master process that receives immigrants and sends the appropriate amount to a random island, since the sizes of the populations are not the same.

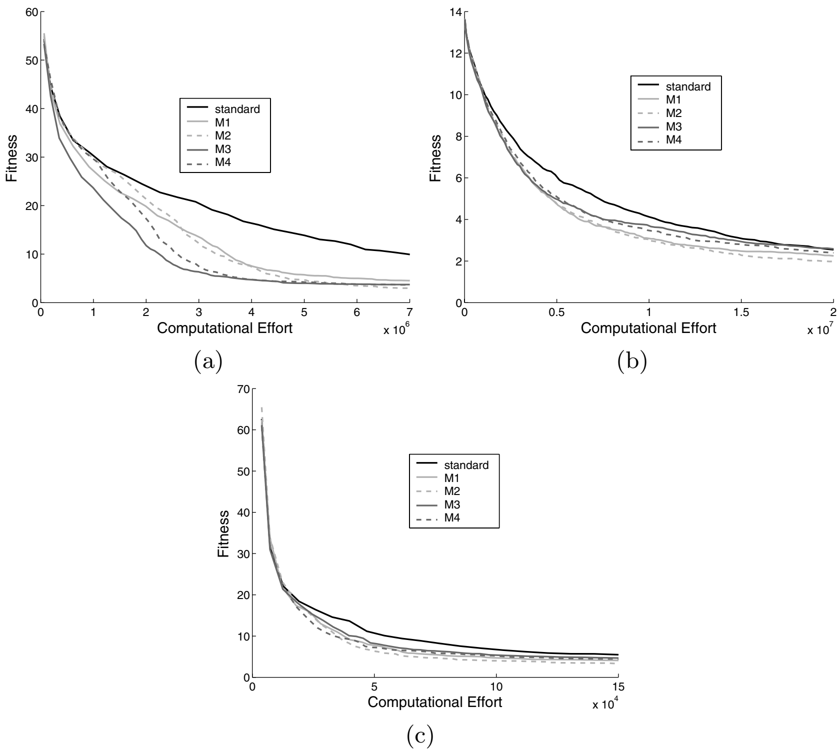


Fig. 8.2. Average fitness results vs. computational effort. Each curve is an average over 100 runs. (a) Artificial ant, (b) even-parity-5, (c) symbolic regression

Evolution of Average Fitness

Figure 8.2 depicts the average fitness of the ensemble of islands for the three benchmark problems as a function of the computational effort. The labels M1, M2, M3, and M4 stand for four slightly different automatic methods for adding individuals when the system detects that the population must grow. The details are not very important, and can be found in [120]. The values are averages over 100 independent executions for each problem and for each method. The thick black curves refers to the *standard* multipopulation model, i.e. the model with fixed-size subpopulations.

For the ant problem (Fig. 8.2 a), we see that all four methods using a dynamic population size give better results with respect to the standard fixed-size-island model. For instance, to reach an average fitness of about 15, the standard distributed model needs a computational effort of about 5.5×10^6, while the same average fitness level is reached by the M1, M2, M3, and M4 variable-island-size methods with an effort that lies approximately between 2×10^6 and 3×10^6.

The even-parity-5 and symbolic regression problems (Figs. 8.2 b, c) show the same general trend, i.e. the variable-size methods are all superior to the standard method but the differences are smaller. Indeed, the standard deviations at the end of the evolution for each problem (not shown in the figures to avoid cluttering them) indicate that, while the differences are significant between the standard model and the ensemble of varying-population-size models for the ant problem, they are not for the other two problems, the differences between the various curves being of the same order as the standard deviation.

The success rate is a good performance indicator when the solution of the problem is known, which is the case here (see Sect. 3.4 for a discussion of this issue). Success rates for each problem are reported in Table 8.1 with their standard deviations. These numbers confirm that, for the ant problem, models with variable-size islands, whatever the method used for suppression and addition of individuals, are better than the standard model. For the even-parity-5 and symbolic regression problems, the results are still favorable to the dynamic island models, since there is always a method among the four that is better than the standard island model. However, for the remaining methods, the differences are within the standard deviations and thus their statistical significance is doubtful.

Evolution of Program Size

Figure 8.3 shows the evolution of the size of the programs in the populations with time. It is easily seen that, in general, the methods M1, M2, M3 and M4, that automatically adjust the population size in the islands offer an easy and implicit means for limiting bloat. This has already been found to be the case for the standard constant-size-island model with respect to standard panmictic genetic programming [60]. Therefore, variable-size populations are a really effective and transparent way for limiting bloat. Now, in the figure it is apparent that method M3 is less effective than M1, M2, and M4 in this respect. But since method M3 is allowed to add individuals to up to twice the original population size, it is clear that this has a negative influence on bloat. However, the other three methods offer performances that are equivalent to

	CE1	CE2	CE3
Standard	0.20(0.04)	0.28(0.04)	0.31(0.05)
M1	0.51(0.05)	0.56(0.05)	0.59(0.05)
M2	0.55(0.05)	0.60(0.05)	0.63(0.05)
M3	0.59(0.05)	0.61(0.05)	0.63(0.05)
M4	0.61(0.05)	0.65(0.05)	0.65(0.05)

(a)

	CE1	CE2	CE3
Standard	0.04(0.02)	0.07(0.03)	0.08(0.03)
M1	0.05(0.02)	0.06(0.02)	0.07(0.03)
M2	0.12(0.03)	0.13(0.03)	0.16(0.04)
M3	0.07(0.03)	0.07(0.03)	0.08(0.03)
M4	0.04(0.02)	0.04(0.02)	0.08(0.03)

(b)

	CE1	CE2	CE3
Standard	0.41(0.05)	0.44(0.05)	0.46(0.05)
M1	0.52(0.05)	0.53(0.05)	0.53(0.05)
M2	0.60(0.05)	0.61(0.05)	0.62(0.05)
M3	0.48(0.05)	0.49(0.05)	0.49(0.05)
M4	0.51(0.05)	0.52(0.05)	0.52(0.05)

(c)

Table 8.1. Success rates at three different values CE1, CE2, and CE3 of the computational effort for the three test problems and the five structured-population models. (a) Artificial ant, (b) even-parity-5, (c) symbolic regression. The standard deviation of each value is included in parentheses

M3, and thus program growth can be controlled best by using one of them. It is clear that, since bloat is not controlled explicitly in the system, it would be possible to add other bloat-reducing techniques. A good candidate could be Poli's "Tarpeian" method, which is theoretically justified and extremely easy to implement [117].

Population Size

In Fig. 8.4 the evolution of the total population size for each problem and each model is plotted. Our first remark is that, except in two cases, where the size is allowed to grow past its initial value, the population size in the adaptive-size models always remains smaller than in the standard model (horizontal lines). This explains why the computational effort required is lower for the same accuracy of the solution. We also see that the evolution of the total

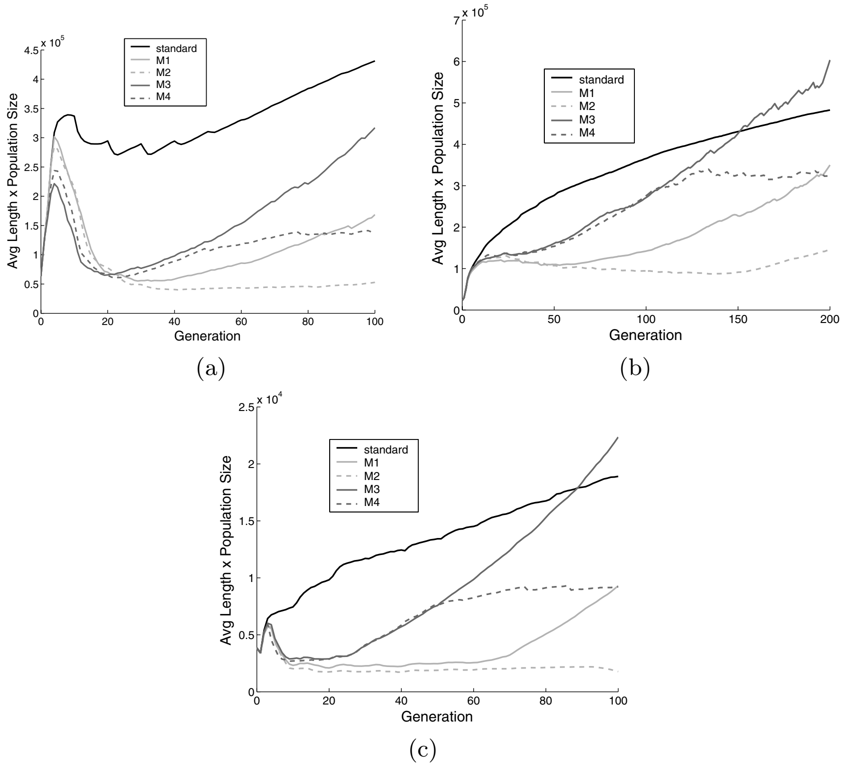


Fig. 8.3. Average population length times total population size is represented as a function of generation number. The averages are over 100 independent runs for each problem and for each model. The thick curves refer to the standard island model, while the other curves refer to the varying-size-population models (see boxes). (a) Artificial ant, (b) even-parity-5, (c) symbolic regression

population size for the four variable-size methods is clearly correlated with the curves of program size shown in Fig. 8.3. This shows empirically that it is possible to fight bloat by automatically adjusting the size of the populations during the run. Considering that the total population size nearly always stays below the constant size of the standard island system, one might fear that diversity would tend to be lost in the variable-size system. However, in [120] it was experimentally shown that this wass not the case, which is reassuring.

It is already empirically well known that multipopulation models are, in general, advantageous with respect to the standard panmictic model, as we have seen in detail in Chap. 3. With varying-size multipopulation systems one finds even better results both in terms of success rates, and from the point of view of the expenditure of effort and program size. But some caution is called for: the findings shown here are empirical, and therefore, they cannot

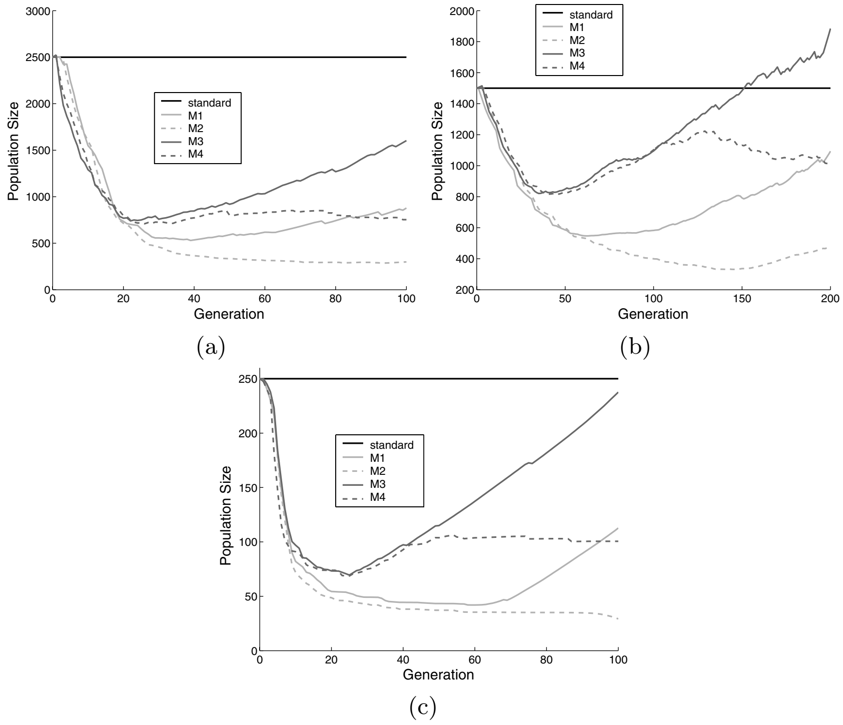


Fig. 8.4. Population size as a function of the generation number. Curves are averaged over 100 runs for each problem and for each model. (a) Artificial ant, (b) even-parity-5, (c) symbolic regression

be extrapolated beyond the simple benchmark problems studied. However, the results are encouraging and show that self-adjusting, variable-size multipopulation systems have something to offer in the way of limiting the computational effort and efficiently solving problems with EAs.

8.2 Nonconventional Cellular Models

In the same way that the island model admits an almost endless series of variations, there are also several structured models that are based on the straight cellular structure of Chaps. 4 and 5 but differ from it in some important respect. I shall present two representative models of this class in this section. The first one is in fact a hybrid variant, while the second uses dynamical neighborhood structures.

8.2.1 The Patchwork Model

This model was introduced in [90]. The main idea was to obtain inspiration from concepts in ecology and population biology. Thus, individuals in the system are mobile agents that live in an ecological niche and interact with their neighborhood by gathering information through sensors, and possibly act on their local environment. Individuals have other attributes such as a maximum life span, mortality, and breeding abilities.

Agents live in a spatial structure that has the shape of a two-dimensional grid. However, each cell in the grid can contain a number of individuals from zero up to a maximum, not just one as in standard cellular EAs. As in cEAs, individuals can migrate to neighboring patches according to some migration policy. Thus, the system combines to some extent the properties of the island and cellular models. Individuals are actually relatively complex agents that act in a two-dimensional virtual world, in a manner similar to many "Alife" models. Individuals can make decisions and adapt their behavior with respect to the environment that they see close to their patch. Decisions are scheduled and executed, resolving any collision that may result from incompatible decisions.

Each agent in the population is composed of two parts: its genome and its *motivation* network. The genome is built from a solution vector, a set of standard deviations, and a parameter vector. The solution vector $\mathbf{x}$ is used to calculate the agent's fitness through an evaluation function that is assumed to be known. The vector of standard deviations σ is used to implement mutation: $\mathbf{x}_{t+1} = \mathbf{x}_t + N(0, \sigma)$, where $N(0, \sigma)$ is a vector of normally distributed independent random numbers with zero mean and standard deviations σ. The parameter vector $\mathbf{p}$ is used in the decision process of the agents, which is a relatively complex process that uses the agent's states, its inputs gathered through sensors from their neighborhood, and the agent's *motivation variables.* In fact, the agents have a set of adaptive behavioral rules that will determine each individual's actions. This is done through the decision-maker that determines and schedules a behavior. The following is a pseudocode of the control structure of the *Patchwork* model (rewritten from [90]):

```
initialize population
generation = 0
evaluate population
while not termination condition do
    for all agents do
        determine agent's behavior pattern
    for each cell do
        for each behavior's pattern do
            resolve conflict
            for for each agent in this cell with this behavior pattern do
        perform action
    remove dead agents from the population
    evaluate population
    generation = generation + 1
end while
```

Each agent also has a reproductive behavior and a limited lifetime. Individuals may die either because they have reached a maximal age or because their mortality, which is anticorrelated with fitness, is high. Offspring also may die when their fitness is low and their number exceeds the available room in a patch. This behavior makes the population size in each patch a variable.

A simplified version of the patchwork system has been tested on some continuous function optimization problems in [90]. Many potential effects of the full-fledged model are absent, but the flavor is the same. The results were good, although the system's complexity is to some extent overkill with respect to the problems chosen. However, of course, many more suitable applications could be found for the system.

In conclusion, the patchwork system is an interesting hybrid structured evolutionary system that is somewhat reminiscent of other spatially extended Alife agent-based systems such as Holland's ECHO system [78] and Langton's SWARM [101]. The aim with these "ecological" simulation environments is to model *complex adaptive systems.* However, their many features and details, while they are an advantage for the appropriate problems, are probably not well suited for straight optimization, where simplicity, speed, and problem-dependent information are easier to obtain within standard heuristics such as simulated annealing, tabu search, ant algorithms, and classical structured EAs augmented with local search.

8.2.2 Dynamic Neighborhoods in Cellular Evolution Strategies

Cellular models, which are based on slow diffusion of individuals through a regular grid have been described and analyzed at length in Chaps. 4 and 5. Weinert et al. [156] have recently proposed a dynamical, self-organizing parallel evolution strategy model inspired by some previous work of Halpern [75]. This model also has relationships to the random-graph-based population structures of Chap. 6, although those were static, while the present model is characterized by a dynamically changing topology.

At the beginning, the population structure is a random graph with N vertices. An adjacency matrix L describes the connectivity of the population, while L_j gives the structure of the local neighborhood of node $j = 1, \ldots, N$. Another matrix M describes the connections of next-nearest neighbors, and it can always be calculated from L.

The two main parts of the algorithm are the following. In the first phase, the neighborhood structure L_j of individual j is altered as a function of the predefined neighborhood dynamics. Connections can be added or deleted according to those rules. After the "topology-variation" phase, an evolution strategy [129] is applied to each individual locally in order to generate λ offspring. Individuals for recombination are chosen from the local neighborhood L_j, and the best offspring replaces individual j. The evolution is synchronous, and the two parts of the algorithm are iterated until a termination criterion has been reached.

Two set of rules were used to vary the neighborhood structure: deterministic and probabilistic rules.

Deterministic Topology Adaptation

In the deterministic case, following Halpern, the new neighborhood depends directly on the neighbor's fitness. Each individual possesses two lists sorted by fitness: a list of neighbors and a list of next-nearest neighbors. There is a *detaching* and an *attaching* rule (Weinert et al. call them "coupling" and "decoupling" rules, respectively). The detaching rule says that the connections to a certain percentage of the least fit neighbors will be suppressed. The attaching rule, on the other hand, says that new connections will be created to a given percentage of the best next-nearest neighbors. During time, this dynamics will tend to favor links between fit individuals, while less fit individuals will be dismissed and could even become isolated.

Probabilistic Topology Adaptation

The idea here is to let the topology of the system self-organize in such a way that each individual always has some neighbor, and such that genetic operations that have been successful have an effect on the subsequent evolution. The detaching rule stipulates that a connection to a neighbor is suppressed if recombination with that neighbor produces an offspring that is worse than the best offspring generated at that site in the current generation.

The attaching rule works as follows: if the number of connections to an individual falls below a given threshold, a given number of new links to randomly chosen individuals are established.

Weinert et al. implemented the system on a parallel multiprocessor machine. They performed a number of experiments, comparing the two structured schemes above with a standard panmictic evolution strategy, and a parallel ES with a complete but static communication graph, respectively. The functions used for the experiments were a 30-dimensional sphere model

$$f(\mathbf{x}) = 100 + \sum_{i=1}^{30} x_i^2,$$

and the 10-dimensional Rastrigin function (see also Sect. 5.4.3)

$$f(\mathbf{x}) = 100 + \sum_{i=1}^{10} (x_i^2 - 10 \cos(2\pi x_i)).$$

Several interesting observations were made in [156]. First, in the self-organized dynamic-lattice algorithms, the deterministic rules tend to produce isolated individuals, while the probabilistic topology update rules reintroduce connections, and thus prevent individuals from becoming permanently detached. With respect to a reference ES model with complete static connectivity between individuals, the dynamic ES with deterministic topology-change rules has a higher convergence speed. On the other hand, the convergence velocity of the dynamic model with probabilistic link updates is similar to that of the standard ES. Overall, it was remarked that a parallel ES with probabilistic communication links between individuals yields a relatively slow strategy with a very good convergence probability. The authors of [156] also measured speedup and found it appreciable but far from linear owing to a relatively costly communication overhead; the main gains are in terms of convergence probability and speed, especially for the multimodal Rastrigin function.

The overall impression is that the kinds of self-organizing structures used there hold some promise for improving evolutionary heuristics that use structured populations. It may be of interest to couple the main ideas presented here with populations structures that are less random, such as the graphs described in the second part of Chap. 6.

A

Implementation Notes

In this book, I have stressed models rather than implementations. This is convenient, since it allows a homogeneous mathematical and structural treatment independent of any material computational constraints. In fact, it is the case that any given structured model can be implemented in several different ways on parallel or distributed hardware. This is one of the reasons why I consider the models to be more fundamental than their implementations. Indeed, it has too often been the case in the past that a model has been designed on the basis of the available hardware, a consideration that tends to limit the model's generality. Thus, structured EAs have often been identified with various kinds of parallel or distributed hardware. The problem with this approach is that while the models have an independent abstract value in themselves, their mapping onto actual computational systems is time- and technology-dependent, since hardware solutions evolve at quite a rapid pace.

On the other hand, when the need for an efficient structured EA arises in any given application field, the chances are that the problem is a difficult one and that it needs large amounts of computing time. It thus becomes important in practice to be able to implement a suitable structured model in such a way that it can make efficient use of the available computational resources. In view of this situation, I think that it is useful to include in this appendix a number of considerations on how to efficiently implement the main structured EA models that have been described, especially those that are at the same time algorithmically interesting from the evolutionary-computation point of view and easy to map onto available computer hardware. The main purpose is thus to give succinct descriptions of those parts of the algorithms that have to take the underlying programming model into account in order to benefit from the performance gains.

A.1 Computing Environment

The preceding paragraph implicitly asks the following question: what is the underlying computational configuration that we should use? There is not a unique answer to that question. A systematic description of parallel and distributed hardware is beyond the scope of these notes. The reader can find more detailed information in the review article [5] and in the references cited therein. However, some considerations will help to understand the choices made in this area.

In the last 20 years several waves, or fads, of high-performance computers have appeared, only to become obsolete a few years later. Thus, single-instruction, multiple-data (SIMD) data-parallel computers such as the Connection Machine have nearly disappeared, except in specialized fields such as signal processing. The same can be said of vector supercomputers, which survive only in some high-performance-computing organizations. Shared-memory parallel computers are still relatively popular, but they do not need to be considered separately since software tools exist for coding parallel programs in message-passing style for these target architectures. The multiple-instruction, multiple-data (MIMD) style of parallel computer architecture is still alive, but mainly in the form of computer farms or networked clusters, where advantage is taken of inexpensive off-the-shelf processor technology together with existing or specialized network infrastructure. It is this kind of computer configuration, which is likely to be available to most research groups, that will be the target of our description, rather than costly esoteric parallel computers.

Having selected the kind of hardware environment, we still have to decide what software tools are going to be used. Here, as well, there are several reasonable alternatives (see, for example, [5]). However, for most practical applications, some form of message-passing plus process-management software will be perfectly acceptable. Again, there are several possibilities. Here we shall describe the use of the MPI software package, a popular process communication and management tool that is sufficiently standardized and easy to obtain and install.

A.1.1 MPI

MPI (Message Passing Interface) is a library of message-passing routines [142] that has been standardized, and is portable across a broad base of computing platforms. The MPI application programmer's interface (API) was defined in the mid-1990s by a large group of people from academia, government, and industry. The interface reflects people's experiences with earlier message-passing libraries, such as PVM. The goal of the group was to develop a single library that could be implemented efficiently on a variety of multiple-processor environments, including true parallel MIMD machines and workstation clusters. This distributed programming environment is rather complete, as it supports process-to-process communication, group communication, setting up and managing communication groups, and interacting with the environment. However, only a handful of primitives are really required to efficiently program distributed EAs, which is an advantage, since the complete environment is rather complex. The interested reader should refer to the offical MPI site [142], where useful information is given. In the following sections I shall outline implementations of the main classes of structured EAs using the MPI library, with an emphasis on island EAs. The program fragments will concentrate on process management and on the communication parts of the algorithm only, assuming that the standard evolutionary operations and their implementation are well known to the reader.

A.2 Implementation of Island EAs

Multipopulation[1] EAs can be implemented very naturally using one subpopulation per processor on networked machines or MIMD parallel computers. One big advantage of island EAs when implemented on distributed/parallel hardware is that their communication and synchronization requirements are rather reduced, which limits the overhead of working in distributed mode. In the following two subsections, we shall see an outline of how these models can be implemented using MPI and standard networked machines. The examples will deal with the genetic programming case. Other EA heuristics such as GAs or ESs can be dealt with in a similar way, and are even easier to implement owing to the fixed-size representation of individual.

A.2.1 Synchronous Islands

The computation of a synchronous-island model for a EA can basically be thought of as a collection of MPI processes, each process representing a population for the specific GP problem. The processes/populations can be evolved in parallel and can exchange information using the MPI primitives. The messages exchanged by these processes are (copies of) groups of GP individuals, and the communication happens through another process called the *master* which runs in parallel with the others and implements a given communication topology.

The code used for all the simulations presented in this book was implemented in C++ and used the MPICH library [104, 105, 138], and it was designed starting from the public-domain GPC++ package [58].

Let `cfg` be a reference to one instance of the GP process; `cfg.NumberOfGenerations` be the maximum number of generations the GP system has to perform and let `freq` be the number of generations elapsed

[1] I thank Leonardo Vanneschi for his help with the implementation of island models. See also [150].

between two successive exchanges of individuals between subpopulations; the algorithm performed by each process/population can be described by the following code fragment (where some details have been omitted and replaced by "..."):

```
for (int gen=1; gen<=cfg.NumberOfGenerations; gen++) {

      pop->generate (*newPop);

      // Delete the old generation and make the new the old one
      if (!cfg.SteadyState)
      {
        delete pop;
        pop=newPop;
      }

      ...

      pop->sortIndividuals();


      if ((gen % freq) == 0) {

        pop->send (0, num_indiv_to_exchange);
        pop->receive (0, num_indiv_to_exchange);

      }

      ...

}
```

The main loop contained in this code is the one that allows the system to execute the iterative EA process. The statement "`pop->generate (*newPop)`" allows the system to create a new population (recorded in the variable `newPop`) by the successive application of the selection, crossover, and mutation operators. After that, the newly generated population replaces the old one, which is deleted from memory by means of the `delete` method. The behavior of this method is the deallocation of the memory that has been allocated to `pop`. Its implementation will not be described here. Of course, the copying of the new population into the old one and the deletion of the old one, are executed only if generational EA is being executed. The case of the steady-state functioning of the algorithm is not described here. After the population has been updated, its individuals are sorted on the basis of their fitness. This sorting algorithm places the best individual contained in the population in the first indexing position (index = 0) and the worst one in the last position. The last step is the commmunication of a group of (good) individuals to another island, which takes place once every `freq` generations. This is realized by the `send` and `receive` methods which use the `MPI_Send` and `MPI_Recv` operations. The parameters of the `send` method are the process number (pid) that the information is sent to (the process with pid = 0 in this case, i.e. the master process) and the number of individuals to be exchanged. Analogously, the parameters of the `receive` method are the sender and the number of individuals to be exchanged. Here is a fragment of code describing the behavior of the `send` method:

```
void MyPopulation::
send (int dest, int tag, int num_indiv_to_exchange) {

  int* global_buffer[num_indiv_to_exchange];

  for (int j = 0; j < num_indiv_to_exchange; j++) {

    NthMyGP(j)->pack (global_buffer[j]);

  }

  MPI_Send (global_buffer, num_indiv_to_exchange,
            MPI_INT, 0, 1, MPI_COMM_WORLD);
}
```

The `for` loop at the beginning of this method allows the method to group into the same data structure all the information that has to be sent (i.e. all the information contained in all the individuals to be exchanged). The `pack` method is used to compress the information that is necessary to reconstruct a single individual. This method uses the `MPI_Pack` primitive. Each packed individual is stored in a line of the `global_buffer` data structure. Recording all the information in a single data structure allows the method to call just one `MPI_Send` primitive in order to send all the requested individuals to the master. This is very important, because it allows the system to save the time overhead derived from the execution of many different send operations, with their associated latency times. The implementation of the `pack` method can be described by the following program fragment:

```
void MyGene::pack(int buffer[])
{

  int mynode=node->value();
  MyGene* current;
```

```
     if (isFunction())
       {

         MPI_Pack (&mynode, 1, MPI_INT, buffer,
                   number_of_nodes*sizeof(int),
                   &current_buffer_position_for_send, MPI_COMM_WORLD);


         for (int n=0; n<containerSize(); n++)
         {
             current=NthMyChild(n);
             current->pack(buffer, number_of_nodes);
         }

       }
     else
       {

         MPI_Pack (&mynode, 1, MPI_INT, buffer,
                   number_of_nodes*sizeof(int),
                   &current_buffer_position_for_send, MPI_COMM_WORLD);


       }
   }
```

Basically, the MPI_Pack primitive is called to insert, into the compressed structure called buffer, the information contained in each node of the tree to be packed. For this reason, when the current node represents a function, the pack method has to be called recursively over all its sons. The code would be simpler for a fixed-size data structure such as a bit string or a vector of real parameters.

The following fragment of code describes the implementation of the receive method:

```
   void MyPopulation::receive (int sender,
                               int num_indiv_to_exchange) {


     int* global_buffer[num_indiv_to_exchange];

     ...

     MPI_Recv (global_buffer,
               num_indiv_to_exchange,
               MPI_INT, sender, tag, MPI_COMM_WORLD, &status);
```

```
    for (int j = 0; j < num_indiv_to_exchange; j++) {

      NthMyGP(containerSize()-1-j)->unpack(global_buffer[j]);

    }

  }
```

Symmetrically to what happens in the send method, a package containing a group of compressed individuals is received from the master using a single MPI_Recv primitive call. Then, the information is uncompressed by the unpack method, which receives as input one line of the global_buffer structure (i.e. one compressed individual) and returns an instance of the MyGP class containing all the information stored in that individual that is needed in order to reconstruct the tree. The received individuals are placed at the bottom of the sorted population, i.e. they replace the worst ones. The containerSize() method returns the total number of individuals that make up each subpopulation. The implementation of the unpack mathod can be described by the following code fragment:

```
void MyGP::unpack (int global_buffer_row[])
{
  int r;
  MPI_Unpack (global_buffer_row, number_of_nodes*sizeof(int),
              &current_buffer_position_for_receive,
              &r, 1, MPI_INT, MPI_COMM_WORLD);

  GPNode& tempfunc=searchForNode (r);
  MyGene& g=*createGene (tempfunc);

  if (g.isFunction()) {
    g.unpack(global_buffer_row);
  }
}
```

Each node of the tree to be reconstructed is uncompressed by one call to the MPI_Unpack primitive. For this reason, the method is called recursively over the sons of each nonterminal node. Each uncompressed node is transformed into a gene by means of the createGene method, whose description can be found in [58].

The last part of this subsection is dedicated to a description of the algorithm for the master process. It is very simple and can be described by the following code fragment:

```
int master (int my_rank, int num_p_total, int num_pop_to_ex,
            int freq) {

  for (int k = 0; k < num_p_total; k++) {

    already_calculated[k] = -1;
  }

  for (int gen=1; gen<=number_of_generations; gen++) {

    if ((gen % freq) == 0) {

      for (int i = 1; i < num_p_total; i++) {

           MPI_Recv (global_buffer,
                     num_pop_to_ex*biggest_column[0],
                     MPI_INT, i, 1, MPI_COMM_WORLD, &status);

           destination = calculate_destination (i, num_p_total,
                                                already_calculated);

           already_calculated[destination] = 1;

           MPI_Send (global_buffer,
                     num_pop_to_ex*biggest_column[0],
                     MPI_INT, destination, 1, MPI_COMM_WORLD);
      }
    }
  }
}
```

The master process keeps track of all the slave processes to which a group of individuals has already been sent, by means of the vector of flags called `already_calculated`. It executes a loop like the one executed by the slave processes, in order to iterate over the generations. At every `freq` generations, it receives a package from each slave process and, for each one of them, it sends the package to a new process, whose pid has been calculated by the `calculate_destination` method. This method is used to implement the particular topology chosen. For instance, implementation of a the random topology can be described by the following code fragment:

```
int calculate_destination (int sender, int num_p_total,
                           int already_calculated[]) {

  int destination;

  destination = (rand() % (num_p_total -1)) + 1;
```

```
    if (already_calculated [destination] != -1) {

      int i = destination;
      while (already_calculated[i] != -1) i= (i+1) % num_p_total;
      destination = i;
    }

    return (destination);
  }
```

In this case, the `calculate_destination` method generates a random number (called `destination` in the code fragment), representing the pid of a slave process. If a package has already been sent to the process whose pid is equal to `destination` during this generation (i.e. if `already_calculated [destination]` equals -1), then the list of the process pids is covered until one process that has not received a package yet is found. That process is returned as the destination of the next package shipment.

A.2.2 Asynchronous Islands

In the case of asynchronous communication among islands, no master process is implemented: an idea of the communication topology is integrated into each process/population. The high-level code executed by each process/population, in the simple case of a ring topology, looks as follows:

```
...
MPI_Request request_send;
MPI_Request request_receive;
MPI_Status status_send;
MPI_Status status_receive;
...
for (int gen=1; gen<=cfg.NumberOfGenerations; gen++) {
      pop->generate (*newPop);
      ...
      if ((gen % freq) == 0) {
          request_send = pop->send (((my_rank+1) % num_p_total),
                                    num_indiv_to_exchange);
          int sender;
          if (my_rank > 0) {
            sender = my_rank - 1;
          } else {
            sender = num_p_total - 1;
          }
          request_receive = pop->receive (sender,
                                          num_indiv_to_exchange);
          vect_of_request[current_position] = request_receive;
```

```
            current_position++;
        }
        for (int z = 0; z < current_position; z++) {
          if (received[z] == 0) {
            MPI_Test(&(vect_of_request[z]), &flag, &status_receive);
            if (flag) {
                for (int j = 0; j < num_indiv_to_exchange; j++) {
                pop->NthMyGP(pop->containerSize()-1-j)->
                                 unpack(global_buffer);
             }
             received[z] = 1;
            }
          }
        }
  ...
  }
```

This time, the send and receive methods are both implemented using nonblocking MPI primitives. For this reason, tests for termination of communications must be performed. This is the reason for the presence of the variables request_send, request_receive, status_send and status_receive, representing the hanging requests of sending and receiving a group of individuals and the status of these requests. In order to make parallel execution more efficient, a vector of requests is used, allowing the system to wait for termination of all the hanging communication operations at one time. As the code fragment above shows, once a receive operation is completed, the content of the message is read and unpacked (via the unpack method, which is exactly the same as in the case of synchronous islands described in Sect. A.2.1). Received individuals replace the worst ones as usual, since the population has been sorted by fitness before executing the communications. The same test for termination is implemented for the status of the send operation (code omitted for simplicity). Tests for termination of the sending and receiving operations are implemented by the MPI_Test function, which returns a variable, called test in the above fragment. If the value of test equals one, the communication operation is completed and the execution can proceed.

The implementation of the send method is slightly different from the case of synchronous communications described in Sect. A.2.1, the main differences being that a value is returned by the method, representing a handle for the request for execution of the current operation, and that the MPI function used is MPI_Isend, i.e. the nonblocking send primitive of MPI. Here is a fragment of code describing the behavior of the send method:

```
MPI_Request MyPopulation::
send (int dest, int tag, int num_indiv_to_exchange) {

  int* global_buffer[num_indiv_to_exchange];

  for (int j = 0; j < num_indiv_to_exchange; j++) {

    NthMyGP(j)->pack (global_buffer[j]);

  }

  MPI_Isend (global_buffer, num_indiv_to_exchange,
            MPI_INT, dest, 1, MPI_COMM_WORLD, &request);

  return(request);
}
```

The send method returns a variable of type MPI_Request, indicating a handle for the request for execution of the current asynchronous send.

Analogously, the implementation of the asynchronous receive method can be described by the following code fragment:

```
MPI_Request MyPopulation::receive (int sender,
                                  int num_indiv_to_exchange) {

  int* global_buffer[num_indiv_to_exchange];
  ...
  MPI_Irecv (global_buffer,
             num_indiv_to_exchange,
             MPI_INT, sender, tag, MPI_COMM_WORLD, &request);

  return(request);
}
```

In this case also, the MPI function used for communication is a nonblocking one (MPI_Irecv) and the value returned by this method is a variable of type MPI_Request, representing a handle for the request for execution of the current receiving operation. Given that it is necessary to unpack the received message only when the receive operation is completed and given that it is more efficient to test all the receive operations at once, and to proceed with the unpacking of the first received message, the unpack operation is called outside the receive function, as shown above. This is not the case for the send function, since the pack operation must be performed before the MPI_Isend.

A.2.3 Summary

In conclusion, straightforward island models are easy to implement on distributed or parallel machines, and have been empirically shown to be efficient problem-solvers both from the algorithmic point of view and in terms of elapsed time. A useful description of how practically to build one such computational cluster is given in [83].

Several evolutionary-computation packages that include the possibility of distributing the load over multiple machines and of communicating between them are publicly available. The following short list, with pointers to relevant web pages, is not meant to be exhaustive; it is just an indication of where to find this kind of software at the time of writing.

- The DREAM Project (various European partners):
  http://www.dcs.napier.ac.uk/%7Ebenp/dream/dream.htm

- Open BEAGLE (Laval University):
  http://beagle.gel.ulaval.ca/index.html

- ECJ (George Mason University):
  http://cs.gmu.edu/ eclab/projects/ecj/

- GALOPPS and lilgp (Michigan State University):
  http://garage.cps.msu.edu/

A.3 Implementation of Lattice Cellular EAs

Regular grid EAs are easy to implement on distributed or parallel systems in the synchronous case. Difficulties may arise only in two cases: when they are to function asynchronously in time and when the computational load is different on different CPUs. These cases are briefly discussed below.

A.3.1 Synchronous Cellular EAs

At the beginning of the 1990s, the hardware of choice for implementing synchronous cEAs was SIMD machines such as the Connection Machine and Maspar. Lattice population structures can be mapped straightforwardly onto such architectures, with one or more individuals (cells) per computational unit. In the frequent case where there are more cells than processors, several cells are assigned to each processor; The processors are multiplexed and deal with each cell sequentially. Today, however, this data-parallel machine architecture has been relegated to specialized computational tasks and is no longer easily available. Nevertheless, using the MPI message-passing paradigm on a standard networked or multiprocessor system, it is easy to implement synchronous cellular EAs simply by domain decomposition. As an example of the technique, we may take the case of a two-dimensional grid (see Chap. 4). Figure A.1 schematically depicts the situation: each processor gets a portion of the grid to work with and does its work independently of the other processors. Each processor in fact implements a standard sequential algorithm to synchronously update each individual in its grid patch and does not need to access the memory of other processors except for the points on the border region. Those points need the values of the individuals nearby in the surrounding patches, as only first neighbors interact. Thus, the only communication needed is the swapping of edge values between neighboring regions, which, since these regions are managed by different processors, requires those processors to send and receive the corresponding messages, and this needs to be done before each time step. Since the workload on all processors is analogous, communication can be synchronously blocking, which maintains the same update order as in the sequential case. If the surface/perimeter ratio of the patches in the domain decomposition is sufficiently high, so as to minimize communication overhead, then performance is good on workstation clusters.

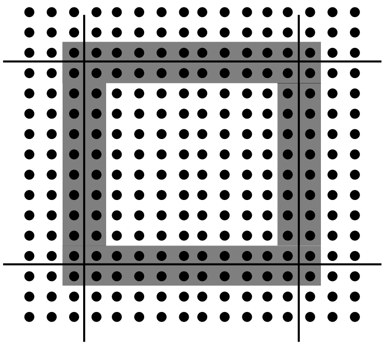


Fig. A.1. Domain decomposition of a two-dimensional lattice. The square domain is broken into square subdomains, subdomains are assigned to different processors, and each processor simultaneously and independently works on its subdomain. Only individuals in the shaded regions need to be communicated between processors

There can be a problem with the aaabove scheme only when the workload of each processor is not the same. This can happen, for instance, in genetic programming, where the evaluation phase, which is usually the more time-consuming, can take widely different times owing to the different size and complexity of difefrent individuals. If the differences are severe, performance will suffer since the computational time per update step will be determined by the slowest processor. This problem can be solved in two ways: either one has to use load-balancing techniques, or one can dispense with strict generational evolution. A clever load-balancing solution, with performance analysis, has been suggested by Spezzano and coworkers and can be found in [56]. The second solution is easier, as it does not require explicit load balancing and can be implemented similarly to asynchronous islands. However, it changes the nature of the algorithm, which can no longer be considered as equivalent to the sequential synchronous version.

A.3.2 Asynchronous Cellular EAs

Asynchronous cellular EAs using random cell update rules such as uniform update or random-sweep update (see Chaps. 4 and 5) are difficult to implement efficiently on standard distributed hardware. Straightforward domain decomposition techniques such as the one described in the previous subsection will not work when one single random cell is updated at each time step. However, variations of those models can be implemented efficiently. For instance, in the random-sweep model, the list of cells to be updated is a random permutation of the n cells in the lattice. Assume that, as before, each processor is assigned one square sublattice to work with. Once the permutation of cells has been generated, all the subpermutations pertaining to each subblock are known. Each processor also has the information about which cell is connected to which in the subblock, i.e. the x–y coordinates of all the cells. Border values can be exchanged between processors as before. By ignoring changes at the boundaries between lists, the algorithm can now be parallelized efficiently, since each processor will work simultaneously and independently on its own list. The only overhead, apart from sparse communication, arises from the setup of the lists in each processor.

A.4 Performance Measures and Speedup

It should be clear that the kind of performance we are interested in here is the time efficiency of the algorithms, and not their quality as problem solvers in the evolutionary sense, an aspect that has been dealt with at length in the rest of the book. The discussion here closely parallels that given in [5]. Computing the speedup of a parallel algorithm is a well-accepted way of measuring its efficiency. Although the use of speedup is very common in the field of deterministic parallel algorithms, it has been adopted in the field of parallel EAs in several flavors, not all of them with a clear meaning. In this section I shall present some concepts related to speedup and discuss their meaning when applied to evaluation of the efficiency of a given parallel/distributed EA.

I shall begin by revisiting the traditional definition of speedup. The well known definition (see e.g. [2]) relates the (worst) execution time of the best sequential version, T_1, to the (worst) execution time of the parallel version of the algorithm on m processors, T_m:

$$S_m = \frac{T_1}{T_m}.$$

With this definition we can distinguish between:

- sublinear speedup $S_m < m$,
- linear speedup $S_m = m$, and
- superlinear speedup $S_m > m$.

The first modification we need to introduce in the standard definition of speedup is to consider average times in the ratio. The reason is that EAs are stochastic algorithms in which a single execution is not statistically significant. This means that we need to average a sufficient number of statistically independent runs in order to have representative time values, and hence

$$S_m = \frac{\overline{T}_1}{\overline{T}_m}.$$

However, even when we use average times, the traditional definition remains unclear in the field of evolutionary algorithms since it makes the assumption that we are aware of the best algorithm to solve the problem. Let us call this the *strong definition* of speedup (Type I in Table A.1). Some practical problems arise with this definition. Firstly, it is difficult, if not impossible, to decide whether or not a sequential EA is the best algorithm, since oftentimes it is the only existing algorithm that has been tried for the problem (e.g.in the case of new applications). Secondly, it is usual when analyzing EAs to study a large set of problems; the strong definition requires the researcher to be aware of the fastest algorithm that solves any of the problems being tackled. This scenario is often not a realistic situation when heuristics are involved.

These reasons have traditionally led researchers to measure the speedup by comparing their own sequential and parallel algorithms. Let us introduce a *weak definition* of speedup (type II in Table A.1) to the extent to which it is possible that a different algorithm exists (probably not an EA) that solves the problem faster in sequential mode. This definition will allow us to compare our parallel EA against well known sequential EAs, therefore studying the speedup without the need to involve nonevolutionary algorithms in the analysis.

I. Strong speedup
II. Weak speedup
    A. Speedup with solution-stop
        1. Versus panmixia
        2. Orthodox
    B. Speedup with predefined effort

Table A.1. Taxonomy of speedup measures

The next important point relating to a weak definition is the stopping criterion. Speedup could be studied by imposing a predefined global number of iterations on both the sequential and the parallel EA. Let us call this a measure of type II.B (Table A.1). In general, this kind of measure is unfair, since it compares two algorithms that are working out solutions of different fitness (quality), thus breaking the fundamental statement that they are "solving" the same problem with the "same" precision. This stopping criterion can be useful in some other situations where, for example, the same effort is allocated to different algorithms to compare their final error, but not when speedup is to be measured. Other researchers [26] have also mentioned these considerations.

Therefore, we need a meaningful and fair termination criterion. The obvious candidate is to stop the algorithms that are being compared when a solution of the same quality has been found, usually an optimal solution. This is called an *orthodox weak* definition, or type II.A.2 (Table A.1).

Let us try to get a deeper understanding of the orthodox weak definition. One important consideration is the composition of the sequential EA. If we were to follow the old-fashioned concept that a "sequential" EA is a "panmictic" EA, we would compare a panmictic (sequential single-population) EA with, for example, a distributed EA of d islands, each one running on a different processor. We call this a *panmixia versus weak* comparison (type II.A.1 in Table A.1). But the algorithm running on a single processor is panmictic in this case, while the d islands that are using d processors represent a distributed migration model whose algorithmic behavior is quite different from that of the panmictic one. This can sometimes produce a very different result for the numerical effort needed to locate the solution, and thus very different search times can be obtained (in general, faster and more accurate search for the distributed version, see Chap. 3). In fact, it could lead to obtaining a superlinear speedup, since a distributed EA running parallel islands can locate a solution more than d times faster than the panmictic one [8], although this need not always be the case.

In order to have a fair and meaningful definition of speedup for parallel EAs, we need to consider exactly the same algorithm (for example, the distributed algorithm with d islands) and then only change the number of processors, from 1 to d, in order to measure the speedup (orthodox weak definition). In any case, the speedup measure should be as fair and as close to the traditional definition of speedup as possible.

Another important point, and an obvious result in parallel computing, is the Amdhal's law [82], namely that adding more processors usually causes a loss of efficiency. Only some models are scalable (more efficient) with an increasing number of processors. Normally, if we add more processors, then we should at the same time increase the population sizes, in order to keep a good computation-to-communication ratio.

We now turn to a striking point. Many authors have analyzed parallel EAs, attending to different criteria, and many have obtained superlinear speedup when using a parallel machine [12, 19, 131]. After having discussed alternative methods to measure the speedup we still have to address one question: is it really possible to obtain superlinear speedup in parallel EAs? The answer to this question is “yes”. In short, the sources for superlinear speedup are (see [3] for more details)

- there is a higher chance of finding an optimum by using more processors, owing to the intrinsically heuristic multipoint nature of parallel EAs;
- splitting a large global population into smaller subpopulations that fit into the processor caches provides faster algorithms than using a single main memory; and
- the operators work on much smaller data structures and they do so in parallel, which is an additional source of speedup.

A.5 A Remark on Pseudorandom Number Generators

I shall end this appendix on technical matters with a short discussion of the possible dangers of generating and using pseudorandom numbers (PRNs) in evolutionary algorithms. The issue has been dealt with in [98] and references therein, to which I shall refer for further experimental details.

EAs are stochastic heuristics that use PRNs rather heavily. Typically, nondeterministic decisions are made when an initial population is generated, when individuals are selected for reproduction, and in the application of most variation operators. Thus, we must be reasonably certain that our methods for generating PRNs are good enough. First, let us briefly review some important concepts in the field. The presentation here is necessarily limited; the reader will find an excellent in-depth study of the issue in Knuth’s classic book [88].

Although one could think of using some natural phenomenon believed to be random, more pragmatically, pseudorandom number generators are used; these are deterministic algorithms that generate strings of numbers that, for most purposes, appear to be random. Given that these sequences cannot be really random, their degree of randomness is gauged by applying a number of statistical tests, such as those described in [88, 97].

In the following the treatment is limited to *uniformly distributed* sequences of pseudorandom numbers, which is the distribution most often used in EAs; however, there are well-known ways for obtaining sequences distributed in other ways starting from uniformly distributed ones.

Random number generators must possess several properties if they are to be useful in lengthy stochastic simulations such as those used in computational physics. The most important properties from this point of view are good results on standard statistical tests for randomness, computational efficiency, a long period, and reproducibility of the sequence.

There exist many ways for generating random numbers on a computer, the most popular one being the *linear congruential generators*. Linear congruential generators are based on the following recurrence formula:

$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0, \ \ m > 0, \ \ 0 < a < m.$$

The value $m > 0$ is called the *modulus*, a is the *multiplier*, and c is an additive constant. Reference [88] describes in great detail how to pick suitable values for these parameters. The sequence clearly has a maximum possible *period* of m, after which it starts to repeat itself. Linear congruential generators are very popular among researchers, and most mathematical software packages include one (or more).

Lagged-Fibonacci generators are also widely used. They are of the form

$$X_n = (X_{n-r} \;\; op \;\; X_{n-p}) \bmod m.$$

The numbers r and p are called *lags*, and there are methods for choosing them appropriately (see [88]). The operator *op* can be one of the following binary operators: addition, subtraction, multiplication, or exclusive or.

A third widespread type of generator is the *linear feedback shift register* (LFSR) generators. A pseudorandom sequence is generated by the linear recursion equation

$$X_n = (c_1 X_{n-1} + c_2 X_{n-2} + \ldots + c_k X_{n-k}) \bmod 2.$$

Linear feedback shift registers are popular generators among physicists and computer engineers. There exist forms of LFSR that are suitable for hardware implementation.

Another popular type of PRN generators is based on cellular automata [79, 148]. It has been shown that these generators can be of very good quality and can easily be implemented in hardware.

In [98] Meysenburg and Foster, using several widely used PRN generators and a set of standard GP test problems, did not find any statistically significant influence of the quality of the generator on the results. On the other hand, Daida et al. [36] found that PRN generators had an effect on the quality of solutions, using another test problem and different generators. On the whole, it seems unlikely that standard good generators will cause any large bias in EA algorithms, although the influence on other aspects of the population such as diversity has not yet been sufficiently investigated. After all, the sequences needed are not very long, as a quick back-of-the-envelope calculation shows. With a population size of 1000, a run that lasts 100 generations, and assuming that all the individuals will be affected by selection, one-point crossover, and mutation, one has of the order of $3 \times 10^3 \times 10^2 = 3 \times 10^5$ random numbers in the sequence. This is a far cry from typical statistical-physics simulations which may require on the order of 10^{10} random numbers. At this scale, correlations and other effects, such as too short a period, that may remain hidden in shorter sequences may become important, and indeed some Monte Carlo simulations in physics have been shown to be heavily biased owing to statistical problems in the random-number sequences. The problem does not seem to be as serious for EAs, as we have seen, unless the generator is a really bad one. However, a wise conclusion for this section could perhaps be the following. While different PRN generators do not seem to have a significant effect on EA computation, we are tending to use larger and larger populations and longer runs, which in turn require more PRNs. Therefore, to be on the safe side, it is a good idea to employ reliable generators, since usually they do not cost more to implement and use. Extra care has to be taken when multiple processes are used, for in this case there is the risk that the same seed may be chosen in different processes, which would lead to spurious correlations. This can be easily avoided in several ways: one possibility is to use the clock time as the seed.

References

1. D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer, Boston, MA, 1987.
2. S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, Englewood Cliffs, New Yersey, 1989.
3. E. Alba. Parallel evolutionary algorithms can achieve superlinear performance. *Information Processing Letters*, 82(1):7–13, April 2002.
4. E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, 2005.
5. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.
6. E. Alba and J. M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN-6*, volume 1917 of *Lecture Notes in Computer Science*, pages 29–38. Springer, Berlin, Heidelberg, New York, 2000.
7. E. Alba and J. M. Troya. Influence of the migration policy in parallel distributed GAs with structured and panmictic populations. *Applied Intelligence*, 12(3):163–181, 2000.
8. E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17:451–465, January 2001.
9. R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.
10. L. A. N. Amaral, A. Scala, M. Barthélemy, and H. E. Stanley. Classes of small-world networks. *Proceedings of the National Academy of Sciences of the USA*, 97(21):11149–11152, 2000.
11. D. Andre and J. R. Koza. Parallel genetic programming: a scalable implementation using the transputer network architecture. In P. Angeline and K. Kinnear, editors, *Advances in Genetic Programming 2*, pages 317–337. MIT Press, Cambridge, MA, 1996.
12. D. Andre and J. R. Koza. A parallel implementation of genetic programming that achieves super-linear performance. *Journal of Information Sciences*, 106(3-4):201–218, 1998.

13. J. Arabas, Z. Michalewicz, and J. Mulawka. A genetic algorithm with varying population size. In *Proceedings of the 1994 IEEE Conference on Evolutionary Computation*, pages 73–78. IEEE Press, Piscataway, NJ, 1994.
14. T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, 1996.
15. T. Bäck, G. Rudolf, and H.-P. Schwefel. Evolutionary programming and evolution strategies: similarities and differences. In D. B. Fogel and W. Atmar, editors, *Proceedings of the Second Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society, San Diego, CA, 1993.
16. S. Baluja. Structure and performance of fine-grain parallelism in genetic search. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 155–162. Morgan Kaufmann, San Francisco, 1993.
17. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming, an Introduction.* Morgan Kaufmann, San Francisco, 1998.
18. A.-L. Barabasi. *Linked.* Plume (Penguin), New York, 2003.
19. T. C. Belding. The distributed genetic algorithm revisited. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 114–121. Morgan Kaufmann, 1995.
20. H. Bersini and V. Detour. Asynchrony induces stability in cellular automata based models. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 382–387. MIT Press, Cambridge, MA, 1994.
21. B. Bollobás. *Random Graphs.* Cambridge University Press, Cambridge, UK, 2001.
22. S. Bornholdt and H. G. Schuster, editors. *Handbook of Graphs and Networks: from the Genome to the Internet.* Wiley-VCH, Weinheim, 2003.
23. E. K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.
24. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer Academic, Boston, 2000.
25. E. Cantú-Paz and D. E. Goldberg. Modeling idealized bounding cases of parallel genetic algorithms. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 456–462. Morgan Kaufmann, San Francisco, 1997.
26. E. Cantú-Paz and D. E. Goldberg. Predicting speedups of idealized bounding cases of parallel genetic algorithms. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 113–120. Morgan Kaufmann, San Francisco, 1997.
27. M. Capcarrère, M. Tomassini, A. Tettamanzi, and M. Sipper. A statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3):255–274, 1999.
28. M. S. Capcarrère, M. Sipper, and M. Tomassini. Two-state, $r = 1$ cellular automaton that classifies density. *Physical Review Letters*, 77(24):4969–4971, 1996.
29. L. L. Cavalli-Sforza, P. Menozzi, and A. Piazza. *The History and Geography of Human Genes.* Princeton University Press, Princeton, NJ, 1994.

30. H. Chen, N. S. Flann, and D. W. Watson. Parallel genetic simulated annealing: a massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.
31. B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems.* Cambridge University Press, Cambridge, UK, 1998.
32. J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Punctuated equilibria: a parallel genetic algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154. Lawrence Erlbaum Associates, Mahwah, NJ, 1987.
33. R. J. Collins and D. R. Jefferson. Selection in massively parallel genetic algorithms. In R. K . Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 249–256. Morgan Kaufmann, San Francisco, 1991.
34. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, second edition, 2001.
35. J. P. Crutchfield, M. Mitchell, and R. Das. Evolutionary design of collective computation in cellular automata. In J. P. Crutchfield and P. Schuster, editors, *Evolutionary Dynamics: Exploring the Interplay of Selection, Accident, Neutrality, and Function*, pages 361–411. Oxford University Press, Oxford, 2003.
36. J. M. Daida, D. S. Ampy, M. Ratanasavetavadhana, H. Li, and O. A. Chaudhri. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Genetic and Evolutionary Conference, GECCO'99*, volume 2, pages 1851–1858. Morgan Kaufmann, San Francisco, 1999.
37. Y. Davidor. A naturally occurring niche & species phenomenon: the model and first results. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 257–263. Morgan Kaufmann, San Francisco, 1991.
38. E. D. de Jong and J. B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–234, 2003.
39. F. Fernández de Vega. *Distributed Genetic Programming Models with Applications to Logic Synthesis on FPGAs.* PhD thesis, Computer Science Department, University of Extremadura, 2001.
40. K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50. Morgan Kaufmann, San Francisco, 1989.
41. B. Dorronsoro, E. Alba, M. Giacobini, and M. Tomassini. The influence of grid shape and asynchronicity on cellular evolutionary algorithms. In *2004 Congress on Evolutionary Computation (CEC 2004)*, pages 2152–2158. IEEE Press, Piscataway, NJ, 2004.
42. B. S. Duncan. Parallel evolutionary programming. In D. B. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 202–208. Evolutionary Programming Society, San Diego, CA, 1993.
43. R. Durrett. *Lecture Notes on Particle Systems and Percolation.* Wadsworth, Belmont, CA, 1988.
44. D. Eby, R. C. Averill, W. F. Punch III, and E. D. Goodman. The optimization of flywheels using an injection island genetic algorithm. In P. J. Bentley, editor,

Evolutionary Design by Computers, pages 167–190. Morgan Kaufmann, San Francisco, 1999.

45. A. E. Eiben, E. Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabaán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 41–50. Springer, Berlin, Heidelberg, New York, 2004.
46. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer, Berlin, Heidelberg, New York, 2003.
47. A.E. Eiben and M. Jelasity. A critical note on experimental research methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)*, pages 582–587. IEEE Press, Piscataway, NJ, 2002.
48. A. Ekárt and S. Z. Németh. Maintaining the diversity of genetic programs. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *Lecture Notes in Computer Science*, pages 162–171. Springer, Berlin, Heidelberg, New York, 2002.
49. N. Eldredge and S. J. Gould. Punctuated equilibria: An alternative to phyletic gradualism. In T. J. M. Schopf, editor, *Models in Paleobiology*, pages 82–115. Freeman Cooper, San Francisco, 1972.
50. F. Fernández, M. Tomassini, and L. Vanneschi. Studying the influence of communication topology and migration on distributed genetic programming. In J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *Lecture Notes in Computer Science*, pages 51–63. Springer, Berlin, Heidelberg, New York, 2001.
51. F. Fernández, M. Tomassini, and L. Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–52, 2003.
52. F. Fernández, M. Tomassini, and L. Vanneschi. Saving computational effort in genetic programming by means of plagues. In *Congress on Evolutionary Computation (CEC'2003)*, pages 2042–2049. IEEE Press, Piscataway, NJ, 2003.
53. F. Fernández, M. Tomassini, and L. Vanneschi. A new technique for dynamic size populations in genetic programming. In *Congress on Evolutionary Computation (CEC'2004)*, pages 486–493. IEEE Press, Piscataway, NJ, 2004.
54. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ, 2000.
55. G. Folino, C. Pizzuti, and G. Spezzano. A cellular genetic programming approach to classification. In W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Genetic and Evolutionary Conference, GECCO'99*, volume 2, pages 1015–1020. Morgan Kaufmann, San Francisco, 1999.
56. G. Folino, C. Pizzuti, and G. Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transactions on Evolutionary Computation*, 7(1):37–53, February 2003.
57. G. Folino, C. Pizzuti, G. Spezzano, M. Tomassini, and L. Vanneschi. Diversity analysis in cellular and multipopulation genetic programming. In *Congress on Evolutionary Computation (CEC'04)*, pages 305–311. IEEE Press, Piscataway, NJ, 2004.

58. A. Fraser and T. Weinbrenner. Genetic programming kernel, version 0.5.2, C++ class library, 1997. A downloadable version, including documentation, can be found at the following url: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/weinbenner/gp.html.
59. H. Fukś. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55(3):2081–2084, 1997.
60. G. Galeano, F. Fernández, M. Tomassini, and L. Vanneschi. Studying the influence of synchronous and asynchronous parallel GP on programs length evolution. In *Congress on Evolutionary Computation (CEC'02)*, pages 1727–1732. IEEE Press, Piscataway, NJ, 2002.
61. M. Giacobini, E. Alba, A. Tettamanzi, and M. Tomassini. Modeling selection intensity for toroidal cellular evolutionary algorithms. In K. Deb et al., editor, *Proceedings of the genetic and evolutionary computation conference GECCO'04*, Lecture Notes in Computer Science, pages 1138–1149. Springer, Berlin, Heidelberg, New York, 2004.
62. M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In E. Cantú-Paz et al., editor, *Proceedings of the genetic and evolutionary computation conference GECCO'03*, Lecture Notes in Computer Science, pages 955–966. Springer, Berlin, Heidelberg, New York, 2003.
63. M. Giacobini, M. Tomassini, and A. Tettamanzi. Modelling selection intensity for linear cellular evolutionary algorithms. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, *6th International Conference on Artificial Evolution, EA 2003*, volume 2936 of *Lecture Notes in Computer Science*, pages 345–356. Springer, Berlin, Heidelberg, New York, 2004.
64. M. Giacobini, M. Tomassini, and A. Tettamanzi. Takeover time curves in random and small-world structured populations. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'05*. ACM Press, 2005. To appear.
65. M. Giacobini, M. Tomassini, A. Tettamanzi, and E.Alba. Selection intensity for cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*. To appear.
66. D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79. Morgan Kaufmann, San Francisco, 1989.
67. D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Francisco, 1991.
68. D. E. Goldberg, K. Deb, and J. Horn. Massively multimodality, deception and genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, PPSN-2*, pages 37–46. North-Holland, Amsterdam, 1992.
69. V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann, San Francisco, 1993.
70. M. Gorges-Schleuter. ASPARAGOS an asynchronous parallel genetic optimisation strategy. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann, San Francisco, 1989.

71. M. Gorges-Schleuter. Evolutionary TSP optimization: an application case study. In L. Plantamura, B. Soucek, and G. Visaggio, editors, *Frontier Decision Support Concepts*, pages 287–318. Wiley, New York, 1994.
72. M. Gorges-Schleuter. An analysis of local selection in evolution strategies. In W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Genetic and Evolutionary Computation Conference, GECCO'99*, volume 1, pages 847–854. Morgan Kaufmann, San Francisco, 1999.
73. J. J Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, Computer Science Department, Nashville, 1981.
74. P.B. Grosso. *Computer Simulation of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model.* PhD thesis, University of Michigan, University Microfilms No. 8520908, 1985.
75. P. Halpern. Evolutionary algorithms on a self-organized, dynamic lattice. In Y. Bar-Yam, editor, *Proceedings of the Second International Conference on Complex Systems.* Perseus, Cambridge, MA, 2001.
76. W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.
77. W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, SFI Studies in the Sciences of Complexity, pages 313–324. Addison-Wesley, Redwood City, CA, 1992.
78. J. H. Holland. *Hidden Order.* Addison-Wesley, Reading, MA, 1995.
79. P. D. Hortensius, R. D. McLeod, and H. C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.
80. B. A. Huberman and N. S. Glance. Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences of the USA*, 90:7716–7718, August 1993.
81. B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.
82. K. Hwang. *Advanced Computer Architecture.* McGraw-Hill, New York, 1993.
83. F. H. Bennet III, J. Koza, J. Shipman, and O. Stiffelman. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'99*, pages 1484–1490. Morgan Kaufmann, San Francisco, 1999.
84. K. A. De Jong, M. A. Potter, and W. M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proceedings of the Seventh ICGA*, pages 338–345. Morgan Kaufmann, San Francisco, 1997.
85. H. Juillé and J. B. Pollack. Coevolving the ideal trainer: application to the discovery of cellular automata rules. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, D. Goldberg, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527. Morgan Kaufmann, San Francisco, 1998.
86. S. A. Kauffman. *The Origins of Order.* Oxford University Press, New York, 1993.

87. S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In *Proceedings of the 22nd ACM Computer Science Conference*, pages 66–73. ACM Press, 1994.
88. D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms.* Addison-Wesley, Reading, MA, third edition, 1998.
89. J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1992.
90. T. Krink, B. H. Mayoh, and Z. Michalewicz. A patchwork model for evolutionary algorithms with structured and variable size populations. In W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Genetic and Evolutionary Computation Conference, GECCO'99*, volume 2, pages 1321–1328. Morgan Kaufmann, San Francisco, 1999.
91. M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.
92. W. B. Langdon and R. Poli. *Foundations of Genetic Programming.* Springer, Berlin, Heidelberg, New York, 2002.
93. S. C. Lin, W. F. Punch, and E. D. Goodman. Coarse-grain parallel genetic algorithms: categorization and a new approach. In *Sixth IEEE Conference on Parallel and Distributed Processing*, pages 28–37. IEEE Press, Piscataway, NJ, 1994.
94. R. H. MacArthur and E. O. Wilson. *The Theory of Island Biogeography.* Princeton University Press, Princeton, NJ, 1967.
95. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, Amsterdam, 1977.
96. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 428–433. Morgan Kaufmann, 1989.
97. G. Marsaglia. A current view of random number generators. In L. Billard, editor, *Computer Sciences and Statistics*, pages 3–10. Elsevier, Amsterdam, 1985.
98. M. M. Meysenburg and J. A. Foster. Random generator quality and GP performance. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. Smith, editors, *Genetic and Evolutionary Conference, GECCO'99*, volume 2, pages 1121–1126. Morgan Kaufmann, San Francisco, 1999.
99. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, Heidelberg, New York, third edition, 1996.
100. Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics.* Springer, Berlin, Heidelberg, New York, second edition, 2004.
101. N. Minar, R. Burkhart, C. Langton, and K. Askenazi. The swarm simulation system: a toolkit for building multi-agent simulations. Technical Report 96-06-042, Santa Fe Institute Working Paper, 1996.
102. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: mechanisms and impediments. *Physica D*, 75:361–391, 1994.
103. M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.
104. MPICH. MPICH – a portable implementation of MPI, 2004. Downloadable versions and documentation can be found at the following url: http://www-unix.mcs.anl.gov/mpi/mpich/.

105. MPIF. MPI: a message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3–4):165–414, 1994.
106. H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann, 1989.
107. H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4):335–360, 1993.
108. H. Mühlenbein, M. Schomish, and J. Born. The parallel genetic algorithm as a function optimizer. *Parallel Computing*, 17:619–632, 1991.
109. M. Munetomo, Y. Takai, and Y. Sato. An efficient migration scheme for subpopulation-based ansynchronously parallel genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 649. Morgan Kaufmann, San Francisco, 1993.
110. J. D. Murray. *Mathematical Biology, I: An Introduction*. Springer, Berlin, Heidelberg, New York, 2002.
111. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
112. N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, Singapore, 1988.
113. L. Pagie and P. Hogeweg. Information integration and red queen dynamics in coevolutionary optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC'2000)*, pages 1260–1267. IEEE Press, Piscataway, NJ, 2000.
114. L. Pagie and M. Mitchell. A comparison of evolutionary and coevolutionary search. Technical Report 02-01-002, Santa Fe Institute Working Paper, 2002.
115. J. Paredis. Coevolving cellular automata: Be aware of the red queen! In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97)*, pages 393–400. Morgan Kaufmann, San Francisco, 1997.
116. C. C. Pettey and M. R. Leuze. A theoretical investigation of a parallel genetic algorithm. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 398–405. Morgan Kaufmann, San Francisco, 1989.
117. R. Poli. A simple but theoretically motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*, volume 2610 of *Lecture Notes in Computer Science*, pages 204–217. Springer, Berlin, Heidelberg, New York, 2003.
118. M. A. Potter and K. A. De Jong. Cooperative coevolution: an architecture for evolving coadapted components. *Evolutionary Computation*, 8(1):1–29, 2000.
119. W. Punch. How effective are multiple populations in genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, D. Goldberg, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 308–313. Morgan Kaufmann, San Francisco, 1998.
120. D. Rochat, M. Tomassini, and L. Vanneschi. Dynamic size populations in distributed genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. I.

van Hemert, and M. Tomassini, editors, *Genetic Programming, Proceedings of EuroGP'2005*, volume 3447 of *Lecture Notes in Computer Science*, pages 50–61. Springer, Berlin, Heidelberg, New York, 2005.

121. J. P. Rosca. Entropy-driven adaptive representation. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32, Tahoe City, California, USA, 1995.
122. J. P. Rosca and D. H. Ballard. Discovery of subroutines in genetic programming. In P. Angeline and K. Kinnear, editors, *Advances in Genetic Programming 2*, pages 177–201. MIT Press, Cambridge, MA, 1996.
123. G. Rudolph. Global optimization by means of distributed evolution strategies. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 209–213. Springer, Berlin, Heidelberg, New York, 1991.
124. G. Rudolph. On takeover times in spatially structured populations: Array and ring. In K. K. Lai, O. Katai, M. Gen, and B. Lin, editors, *Proceedings of the Second Asia-Pacific Conference on Genetic Algorithms and Applications*, pages 144–151. Global-Link Publishing Company, Hong Kong, 2000.
125. G. Rudolph and J. Sprave. A cellular genetic algorithm with self-adjusting acceptance threshold. In *First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 365–372. IEE, London, 1995.
126. J. Sarma and K. A. De Jong. An analysis of the effect of the neighborhood size and shape on local selection algorithms. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 236–244. Springer, Berlin, Heidelberg, New York, 1996.
127. J. Sarma and K. A. De Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 181–186. Morgan Kaufmann, San Francisco, 1997.
128. B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51:123–143, 1999.
129. H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
130. R. Sedgewick. *Algorithms in C++, Graph Algorithms*. Addison-Wesley, Boston, MA, third edition, 2002.
131. R. Shonkwiler. Parallel genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 199–205. Morgan Kaufmann, San Francisco, 1993.
132. M. Sipper. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1996.
133. M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*, volume 1194 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, New York, 1997.
134. M. Sipper. The evolution of parallel cellular machines: toward evolware. *BioSystems*, 42:29–43, 1997.
135. M. Sipper. Programming cellular machines by cellular programming. In D. Mange and M. Tomassini, editors, *Bio-Inspired Computing Machines: Toward Novel Computational Architectures*. Presses Polytechniques et Universitaires Romandes, Lausanne, 1998.

136. M. Sipper and E. Ruppin. Co-evolving architectures for cellular machines. *Physica D*, 99:428–441, 1997.
137. M. Sipper, M. Tomassini, and M. S. Capcarrère. Evolving asynchronous and scalable non-uniform cellular automata. In G. D. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, pages 67–71. Springer-Verlag, Wien, New York, 1997.
138. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI. The Complete Reference.* MIT Press, Cambridge, MA, 1997.
139. G. Spezzano, G. Folino, and C. Pizzuti. CAGE: a tool for parallel genetic programming applications. In J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *Lecture Notes in Computer Science*, pages 51–63. Springer, Berlin, Heidelberg, New York, 2001.
140. P. Spiessens and B. Manderick. A massively parallel genetic algorithm. In R. K. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 279–286. Morgan Kaufmann, San Francisco, 1991.
141. J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. In *Congress on Evolutionary Computation (CEC'99)*, pages 1384–1391. IEEE Press, Piscataway, NJ, 1999.
142. The Message Passing Interface (MPI) Standard. http://www-unix.mcs.anl.gov/mpi/.
143. T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 176–185. Springer, Berlin, Heidelberg, New York, 1991.
144. R. Tanese. Parallel genetic algorithms for a hypercube. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 177–183. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
145. A. Tettamanzi and M. Tomassini. *Soft Computing: Integrating Evolutionary, Neural and Fuzzy Systems.* Springer, Berlin, Heidelberg, New York, 2001.
146. M. Tomassini. The parallel genetic cellular automata: application to global function optimization. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391. Springer-Verlag, Wien, New York, 1993.
147. M. Tomassini, M. Giacobini, and C. Darabos. Evolution of small-world networks of automata for computation. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabaán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 672–681. Springer, Berlin, Heidelberg, New York, 2004.
148. M. Tomassini, M. Sipper, and M. Perrenoud. On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on Computers*, 49(10):1145–1151, 2000.
149. M. Tomassini, L. Vanneschi, F. Fernández, and G. Galeano. Experimental investigation of three distributed genetic programming models. In J. J. Merelo, P. Adamidis, H. G. Beyer, J.-L. Fernández-Villacanas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature, PPSN VII*, volume 2439 of *Lecture*

Notes in Computer Science, pages 641–650. Springer, Berlin, Heidelberg, New York, 2002.

150. L. Vanneschi. *Theory and Practice for Efficient Genetic Programming.* PhD thesis, Computer Science Department, University of Lausanne, 2004.
151. S. Vérel, P. Collard, M. Tomassini, and L. Vanneschi. Fitness landscape analysis of the cellular automata majority problem: view from Olympus. 2005. submitted.
152. P. F. Verhulst. Notice sur la loi que la population suit dans son accroissement. *Correspondances Mathématiques et Physiques*, 10:113–121, 1838.
153. D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness.* Princeton University Press, Princeton, NJ, 1999.
154. D. J. Watts. *Six Degrees: The Science of a Connected Edge.* Norton, London, 2003.
155. D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.
156. K. Weinert, J. Mehnen, and G. Rudolph. Dynamic neighborhood structures in parallel evolution strategies. *Complex Systems*, 13(3):227–243, 2001.
157. D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 658. Morgan Kaufmann, San Francisco, 1993.
158. D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276, 1997.
159. D. Whitley, S. Rana, and R. B. Heckendorn. Island model genetic algorithms and linearly separable problems. In D. Corne and J. L. Shapiro, editors, *Evolutionary Computing: Proceedings of the AISB Workshop*, volume 1305 of *Lecture Notes in Computer Science*, pages 109–125. Springer, Berlin, Heidelberg, New York, 1997.
160. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Index