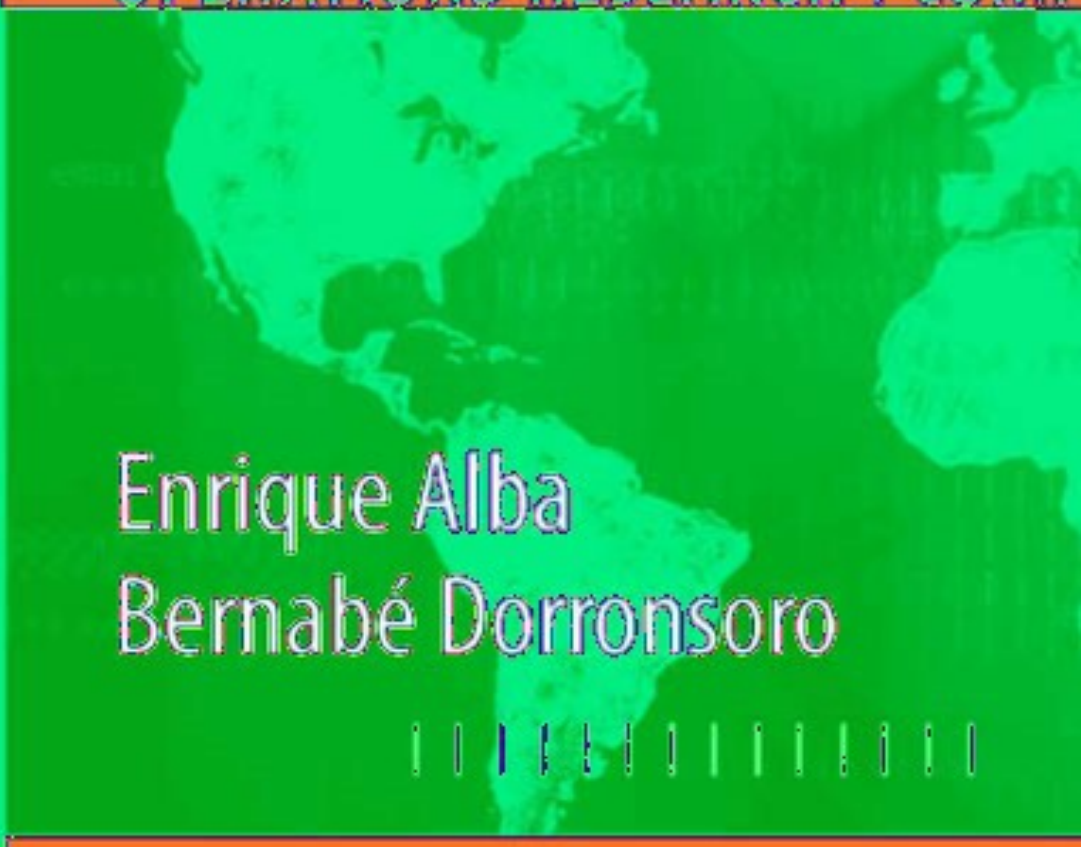
OPERATIONS RESEARCH / COMP



Enrique Alba
Bernabé Dorronsoro

Cellular Ge
Algorithms

Cellular Genetic Algorithms

OPERATIONS RESEARCH/COMPUTER SCIENCE INTERFACES

Professor Ramesh Sharda
Oklahoma State University

Prof. Dr. Stefan Voß
Universität Hamburg

Alba & Dorronsoro / *Cellular Genetic Algorithms*
Bierwirth / *Adaptive Search and the Management of Logistics Systems*
Laguna & González-Velarde / *Computing Tools for Modeling, Optimization and Simulation*
Stilman / *Linguistic Geometry: From Search to Construction*
Sakawa / *Genetic Algorithms and Fuzzy Multiobjective Optimization*
Ribeiro & Hansen / *Essays and Surveys in Metaheuristics*
Holsapple, Jacob & Rao / *Business Modelling: Multidisciplinary Approaches — Economics, Operational and Information Systems Perspectives*
Sleezer, Wentling & Cude/*Human Resource Development And Information Technology: Making Global Connections*
Voß & Woodruff / *Optimization Software Class Libraries*
Upadhyaya et al / *Mobile Computing: Implementing Pervasive Information and Communications Technologies*
Reeves & Rowe / *Genetic Algorithms—Principles and Perspectives: A Guide to GA Theory*
Bhargava & Ye */ Computational Modeling And Problem Solving In The Networked World: Interfaces in Computer Science & Operations Research*
Woodruff */ Network Interdiction And Stochastic Integer Programming*
Anandalingam & Raghavan / *Telecommunications Network Design And Management*
Laguna & Martí / *Scatter Search: Methodology And Implementations In C*
Gosavi/ *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*
Koutsoukis & Mitra / *Decision Modelling And Information Systems: The Information Value Chain*
Milano */ Constraint And Integer Programming: Toward a Unified Methodology*
Wilson & Nuzzolo / *Schedule-Based Dynamic Transit Modeling: Theory and Applications*
Golden, Raghavan & Wasil / *The Next Wave in Computing, Optimization, And Decision Technologies*
Rego & Alidaee/ *Metaheuristics Optimization via Memory and Evolution: Tabu Search and Scatter Search*
Kitamura & Kuwahara / *Simulation Approaches in Transportation Analysis: Recent Advances and Challenges*
Ibaraki, Nonobe & Yagiura / *Metaheuristics: Progress as Real Problem Solvers*
Golumbic & Hartman / *Graph Theory, Combinatorics, and Algorithms: Interdisciplinary Applications*
Raghavan & Anandalingam / *Telecommunications Planning: Innovations in Pricing, Network Design and Management*
Mattfeld / *The Management of Transshipment Terminals: Decision Support for Terminal Operations in Finished Vehicle Supply Chains*
Alba & Martí/ *Metaheuristic Procedures for Training Neural Networks*
Alt, Fu & Golden/ *Perspectives in Operations Research: Papers in honor of Saul Gass' 80th Birthday*
Baker et al/ *Extending the Horizons: Adv. In Computing, Optimization, and Dec. Technologies*
Zeimpekis et al/ *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*
Doerner et al/ *Metaheuristics: Progress in Complex Systems Optimization*
Goel/ *Fleet Telematics: Real-time management & planning of commercial vehicle operations*
Gondran & Minoux/ *Graphs, Dioïds and Semirings: New models and algorithms*

Enrique Alba and Bernabé Dorronsoro

Cellular Genetic Algorithms

Springer

Enrique Alba
Universidad de Málaga
Málaga, Spain
eat@lcc.uma.es

Bernabé Dorronsoro
Universidad de Málaga
Málaga, Spain
bernabe@lcc.uma.es

Series Editors
Ramesh Sharda
Oklahoma State University
Stillwater, Oklahoma, USA

Stefan Voß
Universität Hamburg
Germany

ISBN: 978-0-387-77609-5 e-ISBN: 978-0-387-77610-1
DOI: 10.1007/978-0-387-77610-1

Library of Congress Control Number: 2007942761

© 2008 Springer Science+Business Media, LLC
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now know or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks and similar terms, even if the are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

To my son Enrique and my daughter Ana,
two nice cellular conformations
Enrique Alba

To Patricia and my family
Bernabé Dorronsoro

Preface

This book is the result of an effort to create very efficient search algorithms with a bound cost in their performance and their implementation. Solving optimization and learning problems in academy and industry is of major importance nowadays, not only in computer science, but also in operations research, mathematics, and in almost any domain in daily life: logistics, bioinformatics, economy, telecommunications...

In this context, the research activity on metaheuristic algorithms for solving complex problems is not surprisingly rising in these days. The reason for that is that we are continuously facing new engineering problems which demand more sophisticated solvers. Among the many families of metaheuristics, the application of Evolutionary Algorithms (EAs) has been especially intense during this last decade. These algorithms are usually employed to solve problems of high dimensionality such as constrained optimization tasks, in the presence of noise, having a high degree of epistasis and multimodality.

The behavior of an EA when solving problems is given by the balance they maintain during the search between the *exploration* (diversification) of new solutions and the *exploitation* (intensification) in the space of solutions to the problem. The result of this tradeoff between diversification and intensification of the search is the true key point giving birth to useful tools. With this book, we stress on the proposal of a single family of algorithms which naturally and easily can be tuned to deal with these two forces during their search for a solution. This will endow the reader with a powerful tool to quickly tackle new domains while still using and retaining part of the implementation and knowledge gained with these algorithms.

Traditionally, most EAs work on a single population of candidate solutions (*panmictic* EAs). In this book we explore in depth the benefits of structuring the population by defining neighborhoods on it. The pursued effect is to longer maintain the diversity, thus improving the exploration capabilities of the algorithm, while the exploitation can be easily strengthened by adding local search or many other means. In most cases, these algorithms outperform their equivalent panmictic counterparts in efficiency and accuracy. Among structured EAs, distributed and cellular EAs are the most well known ones. In the book we develop further in the case of cellular EAs (cEAs), where the population

is structured by using the concept of neighborhood much in the way that cellular automata do (hence their name "cellular"), so that individuals can only interact with their closest neighbors in the population. Cellular EAs are not so well known as distributed EAs, but their performance is really impressive and merits a closer look to make the community aware of their power.

This book is targeted to the field of using structured populations in cEAs, especially dealing with cellular genetic algorithms (cGAs), the most popular family of EAs. Specifically, all the work in this book is based on exploring new proposals resulting from extending cGAs to different domains: hybridization, memetics, parallelism, hierarchies, inexpensive self-tuning, probability distributions, or even multi-objective cGAs (wherein the optimization of more than one –usually conflicting– objective is considered). All our proposed new cGAs are compared in this work versus both a canonical cGA and to state-of-the-art algorithms for a plethora of complex optimization problems belonging to the fields of combinatorial, integer programming, continuous, or multi-objective optimization. Our goal is not only to show new models, but also to solve existing problems to the state of the art solutions and beyond, when possible.

The reader can find some amazing starting ideas for numerous new research lines, either from the point of view of new algorithms as from the viewpoint of complex applications. The book can be used for a course on cGAs and also for basic/advanced research in labs. Readers can use it as a whole, and then learn on the many extensions of cGAs to lots of fields, or either can address a given chapter, and get deeper into the details of one algorithm or application.

We are providing a freely available software on the Internet written in Java (JCell) to reproduce our results and to allow readers quickly prototyping new tools for their domain. The java framework Jcell is available at `http://neo.lcc.uma.es/Software/JCell`, and has been developed to be easily extensible and to reduce the learning curve of the user. Since it is written in Java, users can readily deploy it on any computer; the software is at present being used by many international teams with high success.

With this book we think to have put in a single volume a great deal of the knowledge on cellular GAs; the book contains some fundamental theory on cGAs as well as actual competitive algorithms, all sharing some features that will foster their utilization and comprehension in future unseen applications. We really hope that this book could be of any help to the reader in his/her own domain.

Málaga, Spain — *Enrique Alba*
July 2007 — *Bernabé Dorronsoro*

Contents

Part IV Applications of cGAs

Part V Appendix

Part I

Introduction

1

Introduction to Cellular Genetic Algorithms

If one way be better than another, that you may be sure is nature's way.

Aristotle (384 - 322 BC) – Greek Philosopher

Research in exact algorithms, heuristics and metaheuristics for solving combinatorial optimization problems is nowadays highly on the rise. The main advantage of using exact algorithms is the guarantee of finding the global optimum for the problem [202], but the critical disadvantage for real problems (NP-hard) is the exponential growth of the execution time according to the instance size, as well as the unreal constraints they often impose to solve the problem. On the other hand, ad hoc heuristic algorithms tend to be very fast [201], but the solutions obtained are generally not of high quality. In contrast, metaheuristics offer a balance between both [104]: they are generic methods which offer a good solution (even the global optimum sometimes) usually in a moderate run time.

Due to the wide development of computer science in the last years, increasingly harder and more complex problems are being faced continuously. A large number of metaheuristics designed for solving such complex problems exist in the literature [6, 39]. Among them, evolutionary algorithms (EAs) are very popular optimization techniques [25, 31, 33]. They consist in evolving a population of individuals (potential solutions), emulating the biological process found in Nature, so that individuals are improved. This family of techniques apply an iterative and stochastic process on a set of individuals (population), where each individual represents a potential solution to the problem. To measure their aptitude for the problem, the individuals are assigned a fitness value. This value represents the quantitative information used by the algorithm to guide the search. The tradeoff between exploration of new areas of the search space and exploitation of good solutions accomplished by this kind of algorithms is one of the key factors for their high performance with respect to other metaheuristics. This exploration/exploitation balance can be sharpened with some different parameters of the algorithm such as the population used (decentralized or not), the variation operators applied, or the probability of applying them, among others.

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_1, © Springer Science+Business Media, LLC 2008

In this book we focus on cellular genetic algorithms (cGAs), a class of EA with a decentralized population in which the tentative solutions evolve in overlapped neighborhoods, much in the same sense as cellular automata do [245, 259]. Cellular EAs are a promising starting point for the study of new algorithmic models of EAs, as there exist results showing the great utility in promoting the smooth diffusion of the solutions through their populations, and therefore in maintaining the diversity [220]. Moreover, the investigation in cEAs is a rising topic both in theory [98, 99] and practice [26, 84].

In this chapter we first define some important issues for optimization problems. After that, we present a brief introduction to the field of metaheuristics (Sect. 1.2) and, particulary, evolutionary algorithms (in Sect. 1.3). In Sect. 1.4 we describe the two main existing models of decentralized population in EAs: cellular and distributed evolutionary algorithms. Finally, we deal with cellular evolutionary algorithms in depth (Sect. 1.5), paying special attention to the family of cEAs called cellular genetic algorithms –cGAs– in Sect. 1.6. At the end of this chapter, our main conclusions are given (Sect. 1.7).

1.1 Optimization and Advanced Algorithms

In this section, we define some basic notions used along this book. Initially, we give a formal definition of optimization. Assuming minimization (without any loose in generality), we can define an optimization problem as follows:

Definition 1.1 (Optimization). *An optimization problem is formalized after a pair (S, f), where $S \neq \emptyset$ represents the solution space –or search space– of the problem, while f is a quality criterion known as the objective function, defined as:*

$$f : S \rightarrow \mathbb{R} \ . \tag{1.1}$$

Thus, solving an optimization problem consists in finding a set of decision variables values such that the represented solution by these values $\mathbf{i}^* \in S$ satisfies the following inequality:

$$f(\mathbf{i}^*) \leq f(\mathbf{i}), \quad \forall \ \mathbf{i} \in S \ . \tag{1.2}$$

■

Assuming maximization or minimization does not restrict the generality of the results, as we can establish an equivalence between the maximization and minimization problems as [31, 105]:

$$\max\{f(\mathbf{i}) | \mathbf{i} \in S\} \equiv \min\{-f(\mathbf{i}) | \mathbf{i} \in S\} \ . \tag{1.3}$$

According to the domain of S, we can define *binary optimization* ($S \subseteq \mathbb{B}$), *complete* ($S \subseteq \mathbb{N}$), *continuous*($S \subseteq \mathbb{R}$), or *heterogeneous* –or *mixed*– ($S \subseteq \{\mathbb{B} \cup \mathbb{N} \cup \mathbb{R}\}$) problems.

A definition of proximity between different solutions of the search space is necessary for solving an optimization problem. Two solutions are close each other if they belong to the same neighborhood in the search space, and we define the neighborhood of a solution as:

Definition 1.2 (Neighborhood). *Being (S, f) an optimization problem, a neighborhood structure in S can be defined as*

$$N : S \to S \ , \tag{1.4}$$

such that for each solution $\mathbf{i} \in S$ a set $S_i \subseteq S$ is defined. It also holds that if $\mathbf{i}$ is in the neighborhood of $\mathbf{j}$, then $\mathbf{j}$ is also in the neighborhood of $\mathbf{i}$: $\mathbf{j} \in S_i$ iff $\mathbf{i} \in S_j$. ■

In general, in a complex optimization problem the objective function often presents an optimal solution that is an optimum only in its neighborhood, in a determined neighborhood, but which is not optimal if we consider the whole search space. Therefore, a search method can be easily trapped in an optimal value inside a neighborhood, thus giving rise to the concept of *local optimum*:

Definition 1.3 (Local optimum). *Being (S, f) an optimization problem, and $S_{i'} \subseteq S$ the neighborhood of a solution $i' \in S_{i'}$, i' is a local optimum if the next inequality is satisfied (assuming minimization):*

$$f(i') \leq f(i) \ , \qquad \forall i \in S_{i'} \ . \tag{1.5}$$

■

When tackling real life optimization problems, we are usually forced to deal with constraints. In these cases, the area of feasible solutions S is limited to those that satisfy all the constraints.

Definition 1.4 (Optimization with constraints). *Given an optimization problem (S, f), we define $M = \{i \in S | g_k(i) \geq 0 \ \forall \ k \in [1, \ldots, q]\}$ as the region of feasible solutions of the objective function $f : S \to \mathbb{R}$. The functions $g_k : S \to \mathbb{R}$ are called constraints, and these g_k are called differently according to the value taken in $i \in S$:*

$$\begin{aligned} satisfied &:\Leftrightarrow g_k(i) \geq 0 \ , \\ active &:\Leftrightarrow g_k(i) = 0 \ , \\ inactive &:\Leftrightarrow g_k(i) > 0 \ , \ and \\ violated &:\Leftrightarrow g_k(i) < 0 \ . \end{aligned}$$

A global optimization problem is called without constraints iff $M = S$; in other case is it referred to as restricted or with constraints. ■

Our work focuses on the search of global optimal solutions to the considered problem, although due to the difficulty of some problems this criterion can be relaxed. In the next two sections, we present a brief introduction to the field of metaheuristics in general, paying especial attention to evolutionary algorithms.

1.2 Solving Problems Using Metaheuristics

There exist many proposals of algorithmic techniques in the literature, both exact and approximate, for solving optimization problems (see Fig. 1.1 for a basic classification). Exact algorithms guarantee to find the optimal solution for all the finite instances. Generally, since exact methods need exponential computation times when facing large instances of complex problems, NP-hard problems can not be realistically tackled. Therefore, the use of approximate techniques is a rising set of topics in the last decades. In these methods we lose the guarantee of finding a global optimum (often, but not always) in order to find good solutions in a significantly shorter time compared to exact methods.

In the last two decades a new kind of approximate techniques has merged, consisting basically in combining basic ad hoc heuristic methods (approximate techniques with stochastic guided components) in higher level environments in order to explore the search space efficient and effectively. These methods are commonly known as *metaheuristics*. In [39] the reader can find some metaheuristic definitions given by different authors, but in general we can state that metaheuristics are high level strategies having a given *structure* that plans the application of a set of operations (variation operators) to explore high dimensioned and complex search spaces.

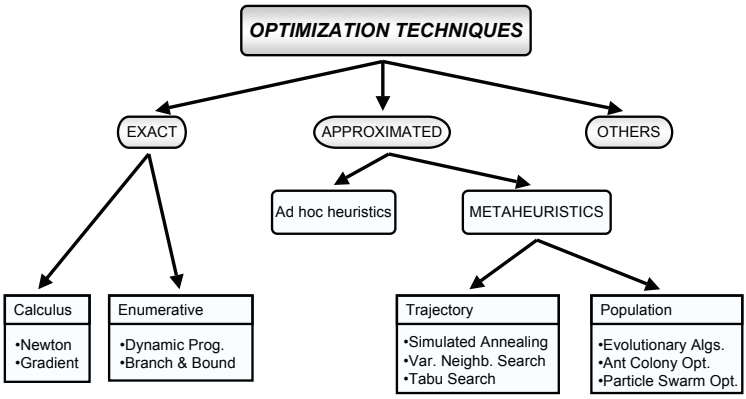


Fig. 1.1. Classification of optimization techniques

Metaheuristics can be classified in many different ways. In [39] a classification is given according to some selected properties which characterize them. So, we can distinguish between metaheuristics inspired by Nature vs. not, population vs. trajectory-based, with a static vs. a dynamic objective function, using one vs. more neighborhood structures, or that memorize previous search steps vs. memoryless techniques. The reader can find a detailed explanation of this classification in [39]. Among the best known metaheuristics, we can find evolutionary algorithms (EA) [33], iterative local Search (ILS) [168], simulated annealing (SA) [145], tabu search (TS) [103], variable neighborhood search (VNS) [183], and ant colony optimization (ACO) [70].

1.3 Evolutionary Algorithms

In the seventies and eighties (with some other punctual works before), several researchers coincided in developing, independently of each other, the idea of implementing algorithms based on the organic evolution model in an attempt to solve adaptive and hard optimization tasks on computers. Nowadays, due to their stockiness and large applicability, and also to the availability of higher computational power (e.g., parallelism), the resulting research field, that of evolutionary computation, receives growing attention from the researchers of many disciplines.

The evolutionary computation framework [33] stands for a wide set of families of techniques for solving the problem of searching optimal values by using computational models, most of them inspired by evolutionary processes (evolutionary algorithms). Evolutionary algorithms (EAs) are population based optimization techniques designed for searching optimal values in complex spaces. They are loosely based on some biological processes that can be appreciated in Nature, like natural selection [57] or genetic inheritance [180] of good traits. Part of the evolution is determined by the natural selection of different individuals competing for resources in the environment. Inevitably, some individuals are better than others. Those that are better are more likely to survive, learn, and propagate their genetic material.

Sexual reproduction allows some shuffling of chromosomes, producing offspring that contain a combination of information from each parent. This is known as the *recombination* operation, which is often referred to as crossover because of the way that biologists have observed strands of chromosomes crossing over during the exchange. Recombination happens in an environment where the selection of the mating pool is largely a function of the fitness of individuals, i.e., how good each individual is at competing in its environment.

As in the biological case, individuals can occasionally mutate. *Mutation* is an important source of diversity for EAs. In an EA, a large amount of diversity is usually introduced at the start of the algorithm by randomizing the genes in the population. The importance of mutation, which introduces further diversity while the algorithm is running, is a matter of debate. Some

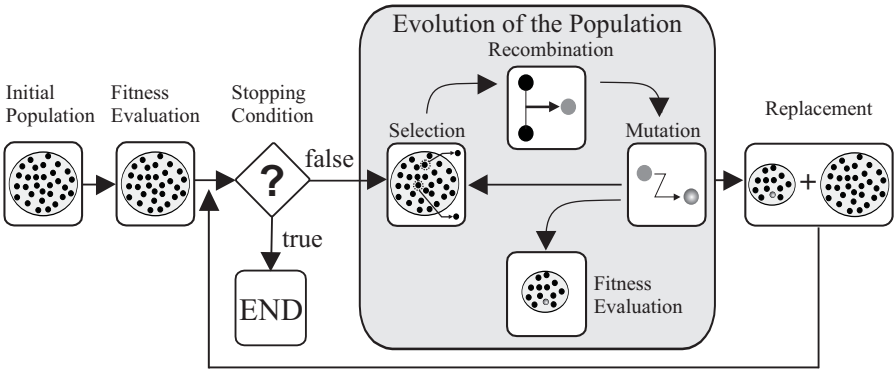


Fig. 1.2. Basic working principles of a typical EA

refer to it as a background operator, simply replacing some of the original diversity which may have been lost, while others view it as playing a dominant role in the evolutionary process (e.g., avoiding getting stuck in local optima).

In Fig. 1.2 we show the functioning of a typical EA. As it can be seen, an EA proceeds in an iterative way by successively evolving the current population of individuals. This evolution is usually a consequence of applying stochastic variation operators such as selection, recombination, and mutation on the population in order to compute a whole generation of new individuals. The initial population is usually generated randomly, although it is also usual to use some seeding technique in order to speed up the search by starting from good quality solutions. A fitness evaluation assigns a value to every individual, which is representative of its suitability to the problem in hand. This evaluation can be performed by an objective function (e.g., a mathematical expression or a computer simulation) or by a subjective opinion, in which the best solutions are selected by an external agent (e.g., expert design of furniture or draws using interactive EAs). The stopping criterion is usually set as reaching a preprogrammed number of iterations of the algorithm, or as finding a solution to the problem (or an approximation to it, if it is known beforehand).

Individuals encode tentative solutions to the problem at hands usually in the form of strings (of binary, decimal, or real numbers), or trees. Every individual has assigned a fitness value as a measure of its adequacy, so better fitness values represent better individuals. This fitness value is used for deciding which individuals are better and which ones are worse. We present in Fig. 1.3 an example of a typical individual structure in EAs.

The chromosome encodes the problem variables in some manner. More than one chromosome could possible exists inside the same individual (e.g., diploid representation like in humans [105]). A single fitness value is allocated to the solution encoded in the chromosome(s) of one single individual after decoding them appropriately (e.g., binary to decimal). Every problem variable

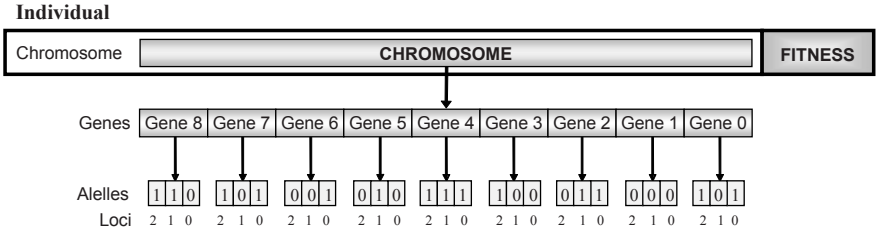


Fig. 1.3. Typical structure of individuals in an EAs

is a gene, and usually (but not forcedly) every gene encodes a value. The individual positions inside every gene are named *loci*, and in general *alleles* are said to be the values stored in every loci.

In Fig. 1.4 we show an example of the application of some specific variation operators in a population composed of 4 individuals for the problem of maximizing $f(x) = x^2$. As it can be seen, the 'String' column in the upper table is the binary codification of the problem variable x using one single gene in one single chromosome in binary. The selection operator chooses the parents with probability equal to the percentage of their fitness values with respect to the sum of the fitness values of all the individuals in the population. The recombination operator (single point crossover) splits the chromosomes of the two individuals into two different parts in a randomly chosen location and joins the parts of the different individuals in order to generate two new offsprings. Finally, the mutation in this example flips the value of a random loci (the first one in this case), in order to introduce some more diversity and hopefully getting a better individual, as it is the case in the figure.

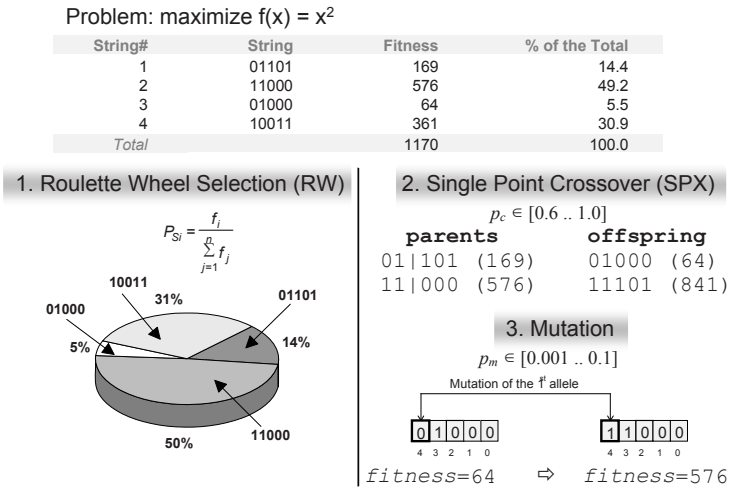


Fig. 1.4. Example of the application of the variation operators in an EA

Algorithm 1.1 Pseudo-code of an Evolutionary Algorithm

```
1. P ← GenerateInitialPopulation();
2. Evaluate(P);
3. while !StopCondition() do
4.       P' ←SelectParents(P);
5.       P' ←ApplyVariationOperators(P');
6.       Evaluate(P');
7.       P ← SelectNewPopulation(P, P');
8. end while
9. Result: The best solution found
```

Now we analyze the functioning of an evolutionary algorithm in detail. Its pseudo-code is shown in Alg. 1.1. As it was said before, evolutionary algorithms work on populations of individuals which are tentative solutions to the problem. The initial population is usually composed by randomly created individuals, although problem knowledge can help creating faster EAs (e.g., by using a greedy initial feeding of solutions). After the generation of the initial population, the fitness value of each individual is computed, and the algorithm starts the reproductive cycle. This step lies in generating a new population through the selection of the parents, their recombination, the mutation of the offsprings obtained and then, their evaluation. These three variation operators are typical of most EAs, specially GAs, although many EA families usually use less (e.g., evolutionary strategies –ES– where no recombination is used) or more additional operators (e.g., decentralized EAs). This new population generated in the reproductive cycle (P') will be used, along with the current population (P), for obtaining the population of individuals for the next generation. The algorithm returns the best solution found during the execution.

There are two kinds of EAs depending on the replacement used, that is, according to the combination between P' and P for the new generation. Thus, being μ the number of individuals of P and λ the number of individuals in P', if the population of the new generation is obtained from the best μ individuals of populations P and P' we have a $(\mu + \lambda)$-EA, whereas if the population of the next generation is composed only by the μ best individuals out of the λ belonging to P', we have a (μ, λ)-EA. In this second case, it is usual that $\mu \leq \lambda$. The "plus" strategy is inherently elitist (the best solution is always preserved) while the "comma" strategy could led to losing the best solution from the population.

The application of EAs to optimization (and learning) problems has been very intense during the last decade [33]. In fact, it is possible to find this kind of algorithm applied to solving complex problems like constrained optimization tasks, problems with a noisy objective function, or problems which have high epistasis (high correlation between the values to optimize) and multimodality [12, 26]. The high complexity and applicability of these algorithms has promoted the emergence of innovative new optimization and search models.

Initially, four kinds of evolutionary algorithms [137] could be clearly differentiated. These four families of algorithms were simultaneously developed by different research groups in the world. Genetic algorithms (GAs) were initially studied by J. H. Holland [133, 134], in Ann Arbor (Michigan), H. J. Bremermann [41] in Berkeley (California), and A. S. Fraser [95] in Sidney (Australia). The evolutionary strategies (ES) were proposed by I. Rechenberg [209, 210] and H.-P. Schwefel [225] in Berlin (Germany), meanwhile the evolutionary programming (EP) was firstly proposed by L. J. Fogel [90] in San Diego (California). Last, the fourth family of algorithms, genetic programming (GP), appeared two decades later, in 1985, as an adaptation of N. Cramer [54] of a genetic algorithm which worked with tree shaped genes, instead of the strings of binary characters traditionally used in GAs, and it is now widely used thanks to the leading works of Koza [153].

Nowadays, the evolutionary algorithms field is growing and evolving itself. An evidence is the number of new families that recently emerged, such as particle swarm optimization (PSO) [40], ant colony optimization (ACO) [70], and estimation of distribution algorithms (EDAs) [158], among others.

1.4 Decentralized Evolutionary Algorithms

Most EAs use a single population (panmixia) of individuals and apply operators on them as a whole (see Fig. 1.5a). In contrast, there also exists some tradition in using structured EAs (where the population is somehow decentralized), which are especially suited for parallel implementation. The use of parallel distributed populations is based on the idea that the isolation of populations enables a higher genetic differentiation [266]. In many cases [27], these algorithms using decentralized populations provide a better sampling of the search space and thus improve both the numerical behavior and the execution time of an equivalent panmictic algorithm. Among the many types of structured EAs, *distributed* and *cellular* algorithms are two popular optimization tools [25, 43, 176, 259] (see Fig. 1.5).

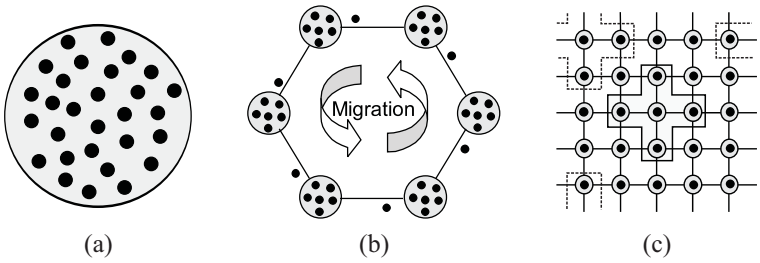


Fig. 1.5. A panmictic EA has all its individuals –black points– in the same population (a). Structuring the population usually leads to distinguishing between distributed (b) and cellular (c) EAs

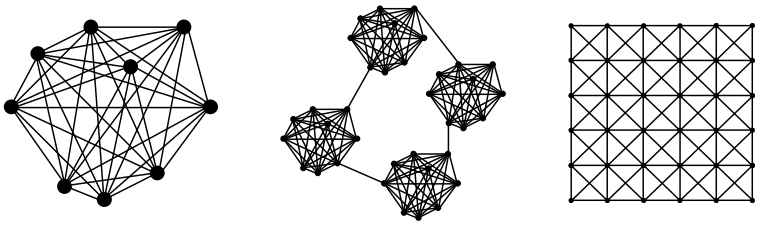


Fig. 1.6. Connectivity graph among individuals for panmictic (a), distributed (b) and cellular (c) EAs

On the one hand, in the case of distributed EAs (dEAs), the population is partitioned in a set of islands in which isolated EAs are executed (see Fig. 1.5b). Sparse exchanges of individuals are performed among these islands with the goal of introducing some diversity into the sub-populations, thus preventing them from getting stuck in local optima.

On the other hand, in a cellular EA (cEA) the concept of (small) *neighborhood* is intensively used; this means that an individual may only interact with its nearby neighbors in the breeding loop (Fig. 1.5c). The overlapped small neighborhoods of cEAs help exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration (diversification), while exploitation (intensification) takes place inside each neighborhood by genetic operations.

If we think on the population of an EA in terms of graphs, being the individuals the vertices of the graph and their relationships the edges, a panmictic EA is a completely connected graph (see Fig. 1.6a). On the other hand, a cEA is a lattice graph, as one individual can only interact with its nearest neighbors (in Fig. 1.6c each individual has 8 neighbors, and the relations between the boundary individuals are not represented for the sake of clarity), whereas a dEA is a partition of the panmictic EA into several smaller EAs, that is, each island is a completely connected graph (very fast convergence), while there exist only a few connections between the islands (as it can be seen in Fig. 1.6b).

These two traditionally decentralized EAs (dEAs and cEAs) are actually two subclasses of the same kind of EA consisting in a set of communicating sub-algorithms. Hence, the actual differences between dEAs and cEAs can be found in the way in which they both structure their populations. In Fig. 1.7, we plot a three-dimensional (3-D) representation of structured algorithms based on the number of sub-populations, the number of individuals in each one, and the degree of interaction among them [5]. As it can be seen, a dEA is composed of a few large sub-populations (having $>> 1$ individuals), loosely connected among them. Conversely, cEAs are made up of a large number of tightly connected sub-populations, each one typically containing a single individual.

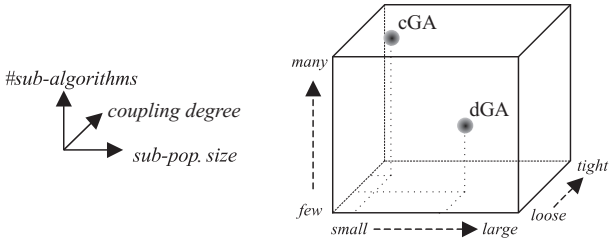


Fig. 1.7. The structured-population evolutionary algorithm cube

This cube can be used to provide a generalized way for classifying structured EAs. However, the points in the cube representing dEA and cEA are just "centroids"; this means we could find or design an algorithm that can hardly be classified as belonging to one of two such classes of structured EAs.

In a structured EA many elementary EAs (grains) exist working on separate sub-populations. Each sub-algorithm includes an additional phase of periodic *communication* with a set of neighboring sub-algorithms located in some topology. This communication usually consists in exchanging a set of individuals or population statistics. All the sub-algorithms were initially thought to perform the same reproductive plan, although there has been a recent trend consisting in executing EAs with distinct parameterizations in each subpopulation, thus performing different searches in the space of solutions in each island. This kind of (usually parallel) EAs are called *heterogeneous* [20].

From this point of view, cellular and distributed EAs only differ in some parameters, and they can be helpful even when run in a monoprocessor [114]. This makes merging them into the same algorithm an interesting option, in order to get a more flexible and efficient algorithm for some kinds of applications [172]. Additionally, any of these EAs can be run in a distributed way (i.e., suited for a workstation cluster), or even on a grid of computers [94] (not to confuse with the "grid" in which individuals evolve in the population).

1.5 Cellular Evolutionary Algorithms

The cellular model simulates the natural evolution from the point of view of the individual. The essential idea of this model is to provide the population of a special structure defined as a connected graph, as we saw in the previous section, in which each vertex is an individual who communicates with his nearest neighbors. Particulary, individuals are conceptually set in a toroidal mesh, and are only allowed to recombine with close individuals. This leads us to a kind of locality known as *isolation by distance*. The set of potential mates of an individual is called its *neighborhood*. It is known [35] that in this kind of algorithm similar individuals can cluster creating *niches*, and these groups operate as if they were separate sub-populations (islands). Anyway, there is no clear borderline between adjacent groups, and close niches can be more easily colonized by competitive niches than in an island model. Simultaneously, farther niches can be affected more slowly.

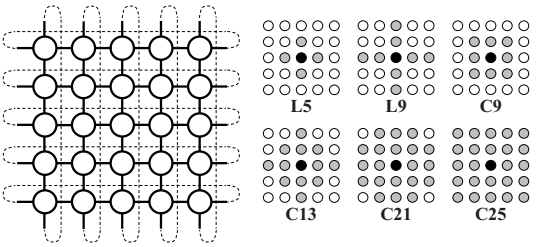


Fig. 1.8. Toroidal population (left) and most typically used neighborhoods in cEAs

In a cEA [246], the population is usually structured in a bidimensional grid of individuals as the one shown on the left of Fig. 1.8, although using this topology does not restrict the scope of the solutions obtained [220]. In it, the boundary individuals of the grid are connected to the individuals located in the opposite borders in the same row/column, depending on the case. This connection is represented in the figure by a dashed line. The acquired effect is a toroidal grid, so that all the individuals have exactly the same number of neighbors. As we said before, the grid used is usually bidimensional, although the number of dimensions can be easily extended to three or more (or reduced).

The neighborhood of a particular point of the grid (where an individual is placed) is defined in terms of the Manhattan distance between the considered point of the grid and the others in the population. Each point of the grid has a neighborhood that overlaps the neighborhoods of nearby individuals; all the neighborhoods have the same size and identical shape. In Fig. 1.8 (on the right-hand side) we can also see the six main neighborhood shapes typically used in cEAs. Please note the names of these neighborhoods: the label Ln (linear) is used for neighborhoods composed by the n nearest neighbors in a given axial direction (north, south, west and east), while the label Cn (compact) is used to designate the neighborhoods containing the $n-1$ nearer individuals to the considered one (in horizontal, vertical, and diagonal directions). The two most commonly used neighborhoods are: (i) L5 [176, 220, 259], also called *Von Neumann* or *NEWS* neighborhood (after *North*, *East*, *West* y *South*); and (ii) C9, also known as *Moore* neighborhood.

In cEAs, the individuals can only interact with their neighbors in the reproductive cycle where the variation operators are applied. This reproductive cycle is executed inside the neighborhood of each individual and, generally, consists in selecting two parents among its neighbors according to a certain criterion, applying the variation operators to them (recombination and mutation for example), and replacing the considered individual by the recently created offspring following a given criterion, for instance, replace if the offspring represents a better solution than the considered individual. In Fig. 1.9 we can see how the reproductive cycle is applied in the neighborhood of an individual in a cEA (the considered neighborhood is NEWS). We must notice that according to the update policy of the population used, we can distinguish between synchronous and asynchronous cEAs (see Sect. 1.5.1); this is also a well-known issue in cellular automata.

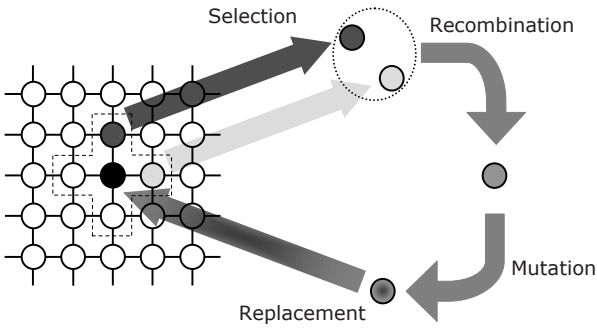


Fig. 1.9. Reproductive cycle of each individual in a cEA

The overlap of the neighborhoods provides an implicit mechanism of migration to the cEA. Since the best solutions spread smoothly through the whole population, genetic diversity in the population is preserved longer than in non structured GAs. This soft dispersion of the best solutions through the population is one of the main issues of the good tradeoff between exploration and exploitation that cEAs perform during the search. It is then easy to think that we could tune this tradeoff (and hence, the genetic diversity level along the evolution) modifying (for instance) the size of the neighborhood used, as the overlap degree between the neighborhoods grows according to the size of the neighborhood. In Fig. 1.10 the overlap degree of two different neighborhoods for the same individuals (drawn in two different grey colors) is shown. On the left part we can see the NEWS neighborhood, while on the right side the C21 neighborhood is shown. The individuals in darker grey color belong to both neighborhoods, while the individuals on the other two lighter grey belong to only one neighborhood: the ones on the lightest grey belong to the light grey individual, and the other grey color is for the individuals belonging to the neighborhood of the dark grey individual.

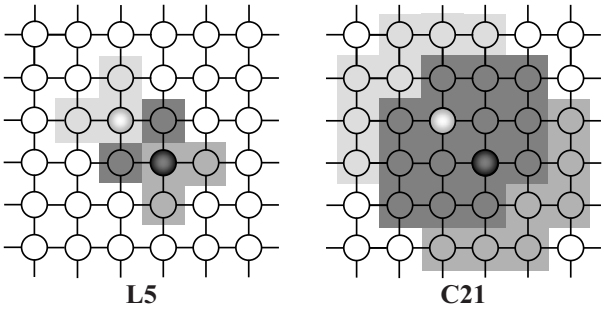


Fig. 1.10. Larger neighborhoods induce a higher level of implicit migration

A cEA can be seen as a cellular automaton (CA) [264] with probabilistic rewritable rules, where the alphabet of the CA is equivalent to the potential set of chromosomes in the search space [245, 259], that is, to the potential number of solutions to the problem. Hence, if we see cEAs as a kind of CA, it is possible to import analytic tools and existing models and proposals from the field of CAs to cEAs in order to better understand these structured EAs and to improve their performance.

Next, we present in Sect. 1.5.1 the two updating methods of individuals that can be applied in cEAs, namely synchronous and asynchronous policies. In Sect. 1.5.2 we introduce a measure for quantitatively characterizing the grid and the neighborhood of a cEA attending to the "radius" concept. Contents of these two sections will allow future discussion and suggest by themselves future research lines.

1.5.1 Synchronous and Asynchronous cEAs

There exist two different kinds of cEAs according to how the reproductive cycle is applied to the individuals. On the one hand, if the cycle is applied to all the individuals simultaneously, the cEA is said to be *synchronous*, as the individuals of the population of the next generation are formally created at the same time, in a concurrent way. On the other hand, if we sequentially update the new individuals of the population with a particular order policy [17] we have an *asynchronous* cEA. An excellent discussion on synchronous and asynchronous cellular automata, relevant for our research, is available in [224]. As we will see through this book, the behavior of an algorithm is really affected by the update policy of the individuals, as well as by other parameters like the size and shape of the neighborhood. Moreover, the visiting order to the individuals for update in the asynchronous case is also a capital issue in the behavior of the algorithm. Next we present the main update policies of individuals in asynchronous cEAs:

- **Line Sweep (LS).** This is the simplest method. It lies in sequentially updating the individuals of the population row by row $(1, \ldots, n)$ in Fig. 1.11.
- **Fixed Random Sweep (FRS).** In this case, the next cell to update is selected with a uniform probability without replacement (that is, it is not possible to visit one cell twice in a generation); this produces a determinate updating $(c_1^j, c_2^k, \ldots, c_n^m)$, where c_q^p means that the cell number p is updated in time q and $(j, k, \ldots, m)$ is a permutation of the n cells. A fixed permutation is used for all the generations.
- **New Random Sweep (NRS).** Similar to FRS, with the difference of using a new random permutation of cells in each generation.
- **Uniform Choice (UC).** In this last method, the next individual to visit is randomly chosen with uniform probability among all the components of the population with replacement (so it is possible to visit one cell more than once in the same generation). This corresponds to a binomial distribution for the updating probability.

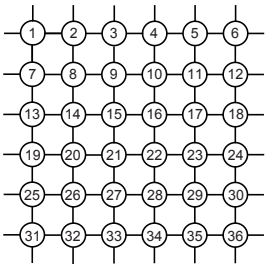


Fig. 1.11. Enumeration of the individuals in the grid

A *time step* (or generation) is defined as the sequential update of n individuals, which corresponds to the update of all the individuals of the population in the cases of the synchronous and LS, FRS and NRS policies, and possibly less than n different individuals for the UC method (as some cells can be updated more than once in the same generation). It is interesting to remark that, with the exception of LS, the other asynchronous update policies are stochastic, which represents an additional source of non determinism besides the one induced by the non-deterministic application of variation operators.

1.5.2 Formal Characterization of the Population in cEAs

In this section we define the characterizing parameters of the population of a cEA. For this purpose, we use the definition of *radius* presented in [26], which is in turn an extension of the concept proposed in [220] which takes into account the non square grids case. In [26] the radius of the grid is supposed to be equal to the dispersion of n^* points in an ellipse centered in $(\overline{x}, \overline{y})$ (Eq. 1.6).

$$\text{rad} = \sqrt{\frac{\sum (x_i - \overline{x})^2 + \sum (y_i - \overline{y})^2}{n^*}} \ , \ \overline{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*} \ , \ \overline{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \ . \quad (1.6)$$

This definition not only characterizes the shape of the grid but also provides a value of the radius for the neighborhood. Although it is called "radius", *rad* measures the dispersion of n^* patterns. This definition always assigns different numeric values to different grid shapes, which differs from the definition proposed in [220], where the measure is the radius of the minimum circle containing the grid, so it could assign identical values to actually different grids.

As was proposed in [220], the relationship between the neighborhood and the grid can be quantified through the tradeoff (or *ratio*) between their radii (Eq. 1.7). Algorithms with a similar ratio show a similar search when using the same selection method, as was concluded in [221].

$$\text{ratio}_{cEA} = \frac{\text{rad}_{\text{Neighborhood}}}{\text{rad}_{\text{Grid}}} \ . \quad (1.7)$$

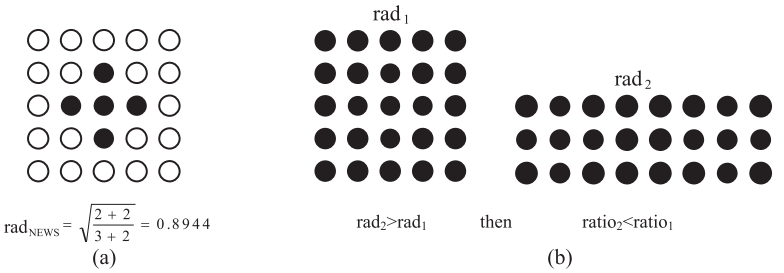


Fig. 1.12. (a) Radius of NEWS neighborhood. (b) Grids $5\times5=25$ and $3\times8\approx25$; the same number of individuals can yield two different ratios

If we have a constant number of individuals ($n = n^\star$, for making fair comparisons), the narrower the grid the higher the value of the topology radius, as it can be seen in Fig. 1.12b. Hence, if we keep constant the size and shape of the neighborhood (for example, using NEWS, Fig. 1.12a), the narrower the population grid the smaller the ratio.

An important feature of manipulating the ratio is that by reducing its value we reduce the intensity of the global selection in the population, so that we are promoting exploration. This is expected to allow a higher diversity in the population and to improve the obtained results for complex problems (as they usually are multimodal and epistatic tasks). Moreover, the search executed inside each neighborhood guides the exploitation of the algorithm.

1.6 Cellular Genetic Algorithms

The main family of EAs under study in this book is that of cellular genetic algorithms. GAs encode the decision variables of a search problem in finite length variable chains of an alphabet of a given cardinality. The resulting strings, which are candidate solutions to the search problem, are called *chromosomes*. Each variable composing the chromosome is a *gene*, while the different values of these genes are named *alleles* (see Fig. 1.3).

Once we encode the problem into one chromosome and define a fitness function for distinguishing promising solutions from the rest, we can start to evolve the population of tentative solutions according to the following steps:

1. **Initialization.** The initial candidate solution population is usually uniformly randomly generated, although it is easy to use problem knowledge or any other kind of information in this step.
2. **Evaluation.** Once we initialized the population, or when a new solution offspring is created, it is necessary to calculate the fitness value of the candidate solutions.
3. **Selection.** Through the selection, we favor the highest fitness value solutions in the search, so a mechanism which encourages the survival of the best individuals is imposed. The main idea of the selection lies in favoring the best solutions against the worst ones, and there exist a plethora of selection procedures proposed for accomplishing this idea [50, 106, 221].

4. **Recombination.** The recombination combines two or more parts from parent solutions for creating new ones (offspring), which are possibly better. A competent performance of the algorithm depends on a well designed recombination mechanism, and we can achieve that in different ways. Ideally, the offspring solution obtained through the recombination is not identical to any of the parents, but contains combined building blocks from both [105].
5. **Mutation.** The mutation randomly modifies a single solution whereas the recombination acts on two or more parent chromosomes. Again, there exist many mutation variants, which usually affect to one or more loci (genes or components) of the individual. In other words, the mutation performs a random jump in the solution space neighborhood of a candidate solution (not to confuse to the structured population neighborhood).
6. **Replacement.** The offspring population created through selection, recombination and mutation, replaces the parent population according to a given criterion; elitist replacement which preserves the best solution along the evolution is a popular criterion.

In Alg. 1.2 we present the pseudo-code of a canonical cGA. As it can be seen, it starts by generating and evaluating an initial population. After that, the already mentioned genetic operators (selection, recombination, mutation, and replacement) are iteratively applied to each individual until the termination condition is met. A key issue characterizing the cellular model is that these genetic operators are applied within the neighborhood of the individuals, so individuals belonging to different neighborhoods are not allowed to interact. The presented pseudo-code in Alg. 1.2 corresponds to a synchronous cGA, as the individuals composing the population of the next generation are stored in an auxiliary population and, when completed, replace in an atomic step the

Algorithm 1.2 Pseudo-code of a canonical cGA

```
1.  proc Evolve(cga)        // Parameters of the algorithm in 'cga'
2.  GenerateInitialPopulation(cga.pop);
3.  Evaluation(cga.pop);
4.  while ! StopCondition() do
5.     for individual ← 1 to cga.popSize do
6.        neighbors ← CalculateNeighborhood(cga,position(individual));
7.        parents ← Selection(neighbors);
8.        offspring ← Recombination(cga.Pc,parents);
9.        offspring ← Mutation(cga.Pm,offspring);
10.       Evaluation(offspring);
11.       Replacement(position(individual),auxiliary_pop,offspring);
12.    end for
13.    cga.pop ← auxiliary_pop;
14. end while
15. end proc Evolve
```

present population. Therefore, in this model all the individuals in the population are updated simultaneously and, equivalently, the creation of individuals is made only from the individuals in the present population (not those previously created during the same iteration). In the asynchronous case, it would not be necessary to use an auxiliary population, since the generated offspring replace the individuals of the population (depending on the criterion used) just when they are visited.

1.7 Conclusions

In this chapter we have presented evolutionary algorithms, which are iterative processes operating on a set of individuals composing a population; each of these individuals represents a potential solution to the problem. This population of individuals evolves due to the application of a set of operators (usually having linear or other light complexity) inspired in biological processes of Nature, such as natural selection and genetic inheritance. As a result, the individuals of the population are improved during the evolution. EAs are very useful tools for solving complex problems, since they work fast in large (and complex) search spaces thanks to their ability to process multiple solutions simultaneously (concept of population). Hence, EAs can follow different search paths simultaneously.

Additionally, it is possible to improve the numerical behavior of the algorithm by structuring the population. The main types of EAs with structured populations are distributed and cellular ones. In this book we focus on the cellular case as it was introduced in this chapter by defining their main components: pseudo-code, characterization of the structured population and the neighborhood, update policies and several other details on cGAs.

2

The State of the Art in Cellular Evolutionary Algorithms

A person who never made a mistake never tried anything new.

Albert Einstein (1879 - 1955) – Physicist

Before starting any new scientific research, it is necessary to know perfectly well the existing contributions to the considered field in the literature. This documentation step is basic for the right development of science, as it provides important knowledge of the working area, allowing us to take advantage of the contributions of others authors, and thus avoiding the development of low interest works as, for example, studies tackled before by other researchers. Hence, in this chapter we present a wide exploration of the state of art in cellular evolutionary algorithms, including and classifying some of the main existing publications related to this field.

The chapter structure is detailed next. We start with Sect. 2.1 explaining the first appeared models of cEAs in the literature. In Sect. 2.2 the main theoretical studies developed in cEAs are summed up, whereas Sect. 2.3 compiles some of the most relevant works where empiric studies of the functioning of cEAs have been carried out, and also comparisons to other models. A summary of the most important works contributing with algorithmic improvements to the cEAs field is shown in Sect. 2.4. Section 2.5 presents some works with high repercussion in the field of parallel cEAs. Finally, at the end of the chapter we summarize all this and mention some open research lines.

2.1 Cellular EAs: a New Algorithmic Model

The cellular evolutionary algorithms were initially designed for working in massive parallel machines, composed of many processors executing simultaneously the same instructions on different data (SIMD machines –Single Instruction Multiple Data) [88]. In the simplest case, the executed cEAs in this sort of machines used a single large population and assigned an only single individual to each processor. In order to avoid a high overload in communications, the mating of the individuals was restricted to the closer individuals (that is, the ones belonging to their neighborhood).

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_2, © Springer Science+Business Media, LLC 2008

In 1976, Bethke [38] made the theoretical study of a GA on a SIMD parallel machine, analyzing the efficiency of the processing capacity use. He concluded that the maximum efficiency is obtained when much more expensive fitness functions than evolution operations are evaluated, a typical case for many applications.

The first cGA model known is the one proposed by Robertson in 1987 [216], implemented on a CM1 computer. It was a model where all the steps of the algorithm (parent selection, replacement, recombination, and mutation) were executed in parallel. This model obtained good results, with an execution time independent from the population size.

A year later, Mühlenbein, Gorges Schleuter, and Kramer published a work [186] where a cGA on massive parallel machines for the TSP problem was proposed. An important characteristic of this cGA was the incorporation of a local search step for improving the generated solutions by the recombination and mutation operators. Therefore, it is considered the first published hybrid cGA.

After these two first works some cGAs appeared in a few years. They were named in terms of the *pollination plants* [105], *parallel individual* [130], *diffusion* [33], *fine grained* [176], *massively parallel* [129, 232] or *local selection* [116] models. The term *cellular GA* was not used until 1993, when Whitley proposed it for the first time in a work where a cellular automaton model was applied on a genetic algorithm [259].

All these cGAs were initially designed for working on massively parallel machines, although, due to the fast loss of popularity suffered by this kind of machines, the model was adopted later for also functioning on mono-processor machines, without any relation to parallelism at all. In fact, since the emergence of cEAs, there were implementations in secuencial environments [59], in *transputers* nets [115], or in parallel distributed environments [177]. This issue should be clear, as many researchers still think about the equivalence between massively parallel EAs and cellular EAs, what represents a wrong connection: cEAs are simply a different class of EAs, as memetic algorithms [184], estimation of distribution algorithms [158, 187], or particle swarm optimization algorithms [40] are.

Since the cGAs appeared, there have been many contributions published in this field. In Table 2.1 we summarize some of the most important ones, which are commented in Sects. 2.2 to 2.4.

2.2 The Research in the Theory of the Cellular Models

The number of existing works in the area of the cGAs theory is too low. This is probably due to the difficulty of deducing generic tests in an area where so many possibilities of implementation exist. Maybe, another reason for this lack is the generalized belief that cGAs model in a more accurate way the populations of nature with respect to the islands model or sequential GAs. Apart from the reason of this lack of theory, more research is necessary in this area.

Table 2.1. Brief summary of the main contributions to the cEAs field

Reference	Contributions
(Hillis, 1990) [129]	cGA with two co-evolutionary populations
(Collins & Jefferson, 1991) [50]	Study of the influence of different selection methods
(Gorges-Schleuter, 1992) [116]	cGA with a migration mechanism
(Gordon et al., 1992) [112]	cGA with a migration mechanism
(White & Pettey, 1993) [258]	Study of different ways of applying the selection method
(Rudolph & Sprave, 1995) [219]	Use of a self-adaptive acceptation threshold
(Sarma & De Jong, 1996) [220]	First theoretical study on the selection pressure in cGAs
(Sarma & De Jong, 1997) [221]	Study on the influence of different selection methods
(Sipper, 1997) [230]	Co-evolutionary cGA
(Folino et al., 1998) [91]	A cGA with a local search step for SAT
(Laumanns et al., 1998) [159]	Prey/predator algorithm for the multi-objective domain
(Gordon et al., 1999) [111]	Heterogeneous cGA: different parameterization in each cell
(Gorges-Schleuter, 1999) [119]	Comparison of panmictic versus cellular ESs
(Kirley et al., 1999) [149]	A cGA allowing empty cells (without individuals)
(Ku, Mak & Siu, 1999) [157]	A cGA with local search for training recurrent neural networks
(Sprave, 1999) [234]	Hypergraphs based model for characterizing cEAs
(Alba & Troya, 2000) [26]	Influence of the ratio on the exploration/exploitation
(Kirley & Green, 2000) [148]	A cGA applied to the continuous optimization domain
(Lee, Park & Kim, 2000) [160]	A cGA with migrations
(Rudolph, 2000) [218]	Takeover in cGAs with ring and toroidal population
(Krink et al., 2000) [215]	A cGA with disasters
(Thomsen & Kirley, 2000) [244]	RBGA: cGA based in religions
(Krink et al., 2001) [155]	[215] with a sand bag model for disasters frequency
(Llor & Garrell, 2001) [167]	GALE: cGA for data mining. Empty cells are allowed
(Kirley, 2002) [147]	CGAD: cGA with disasters
(Alba & Dorronsoro, 2003) [10, 12]	Proposal of cGAs with adaptive population
(Li & Sutherland, 2002) [165]	Prey/predator algorithm for continuous optimization
(Giacobini et al., 2003) [100]	Takeover in asynchronous cGAs with ring population
(Giacobini et al., 2003) [99]	Selection pressure study in cGAs with ring population
(Li, 2003) [164]	Prey/predator algorithm for the multi-objective domain
(Alba et al., 2004) [9]	Comparison between cGAs and other EAs
(Alba & Dorronsoro, 2004) [11]	Some hybrid cGAs for VRP
(Dorronsoro et al., 2004) [17, 74]	Comparison between synchronous and asynchronous cGAs
(Giacobini et al., 2004) [98]	Selection pressure in asynchronous cGAs with toroidal pop.
(Alba et al., 2005) [7]	Asynchronous cGAs with adaptive populations
(Alba et al., 2005) [8, 18]	cMOGA: first orthodox multi-objective cGA
(Alba et al., 2005) [15]	First memetic cGA (cMA); applied on SAT
(Alba & Saucedo, 2005) [24]	Comparison between cGA and panmictic GAs
(Dick, 2005) [68]	A cGA with ring population as a method for preserving niches
(Dick, 2005) [69]	Modelling the genetic evolution in cGAs (ring population)
(Giacobini et al., 2005) [102]	Modelling cGAs with square and rectangular populations
(Giacobini et al., 2005) [101]	Modelling cGAs with small world topology populations
(Alba & Dorronsoro, 2006) [13]	Hybrid cGA which improves the state of art in VRP
(Alba et al., 2006) [23]	First EDA with population structured in a cellular way
(Dorronsoro & Alba, 2006) [73]	A cGA for the numerical optimization domain
(Grimme & Schmitt, 2006) [125]	Prey/predator algorithm for multi-objective domain
(Ishibuchi et al., 2006) [138]	A cGA with distinct neighborhoods for selection and crossover
(Luo & Liu, 2006) [171]	A cGA designed for GPUs
(Luna et al., 2006) [169, 170]	Comparison of cMOGA versus other MO algoritms
(Nebro et al., 2006) [193, 194]	MOCell: a new orthodox MO cGA
(Payne & Eppstein, 2006) [203]	Study of the emergence matching topology in cGAs
(Janson et al., 2006) [139]	HcGAs: cGAs with hierarchical population
(Simoncini et al., 2006) [229]	A new anisotropic selection operator for cGAs
(Xhafa, 2006) [268]	A cMA for task scheduling in *grid computing*
(Alba & Dorronsoro, 2007) [14]	Wide study of a memetic cGA on VRP
(Xhafa et al., 2007) [269]	A cMA versus other GAs for batch task scheduling in grids
(Nebro et al., 2007) [195]	Improvement of MOCell [193, 194]
(jMetal, 2007) [80]	Multi-objective algorithms library (including MOCell)
(Dorronsoro et al., 2007) [75]	Parallel hybrid cGA for large instances of VRP

In the following subsections, we summarize the main existing contributions to the theory of cEAs. Thus, Sect. 2.2.1 presents some studies for theoretically modelling the behavior of cEAs, while Sect. 2.2.2 shows a summary of the main works attempting to characterize the behavior of the cEAs according to the neighborhood to population ratio.

2.2.1 Characterizing the Behavior of cEAs

An easy way for characterizing the search performed by a cEA lies in using the selection pressure, which is a measure of the diffusion speed of the good solutions through the population. In order to deepen into the knowledge of the functioning of cEAs, some theoretical works compare the algorithms according to the selection pressure showed, and in some cases even intend to mathematically model its behavior.

Sarma and De Jong made in [220, 221] some theoretical studies about the selection pressure induced by cGAs with different selection operators and neighborhood sizes and shapes. For studying the effect on the size of neighborhood in the selection pressure, they proposed in [220] a definition of the radius of the neighborhood as a measure of its size. Moreover, they observed the same effect when changing the size of the population, so they proposed a new measure called *ratio*, defined as the relationship between the radii of the neighborhood and the population. Sarma and De Jong discovered that this ratio is a key issue for controlling the selection pressure of the algorithm. Therefore, two algorithms with different population sizes and neighborhoods but with the same ratio value have a similar selection pressure. Finally, they proposed the use of a logistic function (parameterized with one variable) for approaching the curve of the selection pressure presented by cGAs. This function is based on the family of logistic curves which was demonstrated in a previous work [106] that works in the panmictic case. The model proposed seemed to be a good approach for cGAs with square populations, but later it was demonstrated that this model have some deficiencies when using rectangular populations (see Chap. 4 for more information).

Three years later, Sprave proposed in [234] a unified description of any kind of EA with both structured and non-structured populations based on the concept of *hypergraph*. A hypergraph is an extension of a canonical graph, where the concept of the edges is generalized: instead of the union of a pair of vertex they become unions of subsets of vertexes. Using the concept of hypergraph, Sprave developed in [234] a method for estimating the growth curve of the selection pressure of a GA. This method is based on calculating the diameter of the structure of the population and the probability of the distribution induced by the selection operator.

Gorges-Schleuter studied in [119] the growth curves for a diffusion (cellular) model of Evolutionary Strategy (ES) with populations structured in toroidal or ring shapes. In her studies, she observed that the diffusion model of ES (both the toroidal and the ring model) has a lower selection pressure

than the equivalent ES with panmictic population. Moreover, comparing the two diffusion models, she concluded that, using the same neighborhood size, structuring the population in a ring shape allows a lower selection pressure than when using a toroidal population.

In this work, the differences in the behavior of the algorithm with distinct selection schemes were also analyzed. Particulary, two cases were studied: the one in which it is forced that the actual individual is one of the parents, or the case in which the two parents are selected with the same selection method. It was also studied the effect of allowing one single individual to be chosen as both parents (self-matching) or not. In fact, if we force the actual individual to be one of the parents, the error induced by stochastic sampling is reduced, and also the changes in the individuals are more gradual due to one survived offspring will replace that parent.

In [100], Giacobini et al. proposed quantitative models for estimating the *takeover* time (the time for colonizing the population by copies of the best individual only under the selection effects) for synchronous and asynchronous cGAs with structured population in ring shape (one dimension), and using a neighborhood composed by the two nearest individuals to the considered one. This work was later extended in [98, 99] in order to find accurate mathematical models for fixing the selection pressure curves of synchronous and asynchronous cGAs. In these works, the population is structured in a bidimensional grid, but it is forced to be square. In [102], the same authors proposed some probabilistic recurrences for modelling the behavior of the selection pressure of some synchronous and asynchronous cGAs with square, linear (ring), and toroidal population for two different selection schemes. They also studied the case of a rectangular population for synchronous and asynchronous cGAs, but in this case they only validated the experiments on one selection scheme. In Chap. 4 it is demonstrated that the model is not completely satisfactory when other selection schemes different from the one studied in [102] are tested.

Giacobini, Tomassini, and Tettamanzi proposed in [101] some mathematical models for approaching the growth curve of cGAs working on populations where the topology is defined as a random graph or as *small world graphs*, where the distance between any two individuals is, in general, much lower than in the case of the most commonly regular grids used (they are neither regular nor completely irregular [197] graphs).

Recently, in [229], Simoncini et al. proposed a new selection operator for cGAs called *anisotropic selection* for tuning the selection pressure of the algorithm. This new method lies in allowing the selection of the individuals of the neighborhood with different probabilities according to their location. In this way, the authors promote the emergence of niches in the population. In this work the selection pressure of the algorithm on different shapes of population was studied, but the comparison between the new algorithm and the canonical cGA is missed.

Finally, a new contribution to the field of theory of cGAs was presented in [72] through the development of a more accurate mathematical equation than the existing ones for modelling the selection pressure curves of cGAs with rectangular and square populations. The proposed model was demonstrated to be valid for different selection methods.

2.2.2 The Influence of the Ratio

In the literature, there exist results (as [189] for the case of large instances of the TSP problem, or [35, 84, 114] for function optimization) that suggest, but do not analyze, that the shape of the grid of the population really influences in the quality of the search performed by the algorithm. However, Sarma and De Jong defined in [220, 221] the concept of ratio. As it was introduced in Sect. 2.2.1, this feature is highly interesting because algorithms with similar ratio values show a similar behavior in the search.

It was not until year 2000 when Alba and Troya [26] published the first quantitative study of the improvement obtained in the efficiency of a cGA when using non square grids. In this work, the behavior of some cGAs with different shapes of grids on some problems was analyzed, concluding that the use of non square grids promotes a very efficient behavior on the algorithms. Moreover, Alba and Troya redefined in [26] the concept of *radius* as the dispersion of a set of patterns, since the definition by Sarma and De Jong [220] can assign the same numeric value to different neighborhoods, which is undesirable. Finally, we can find another really important contribution in [26], consisting in changing the shape of the population in an specific time step of the execution for modifying the tradeoff between exploration and exploitation applied by the algorithm on the search space. So, the authors take advantage of the different behavior showed by the cGAs with distinct population shapes, and in a very easy way (free of computational load) they change the behavior of the algorithm, in the middle of the execution, promoting the exploration of the search space from a local exploiting step or vice versa.

As a contribution to this area, Dorronsoro and Alba developed in [12] a new adaptive model in which the shape of the population is automatically changed (and, therefore, the ratio value) for regulating the balance between exploration and exploitation performed by the algorithm. Different versions of this new adaptive algorithm were compared to static ratio algorithms, and it improved all of them in all the cases (see Chap. 6).

2.3 Empirical Studies on the Behavior of cEAs

In this section we present some important works for analyzing the behavior of cEAs, like the evolution process of the individuals in the population, or the algorithm complexity according to the operators used.

Collins and Jefferson characterize in their work [50] the difference between the panmictic GAs and the cGAs according to some factors, as the diversity of the genotype and phenotype, the speed of convergence, or the stockiness of the algorithm, concluding that the local matching performed in the cGAs "... *is more appropriate for the artificial evolution ...*" than the GAs with panmictic population. The authors demonstrate in this work that, for a particular problem with two optima, a panmictic GA rarely finds the two solutions, meanwhile the cGA generally finds both solutions. The reason is that, thanks to the slow diffusion of the best solutions produced by the cGA, the diversity is kept for longer in the population, forming some small *niches* in it, or groups of similar individuals, representing different searching areas of the algorithm. This work inspired some other modern works where cGAs are used as methods for finding multiple optimal solutions to problems [68].

Davidor developed in [58] a study about a cGA with a bidimensional grid and a neighborhood with eight individuals. In this study, the proportional selection was used (according to the fitness value) for both parents, creating two offsprings in the recombination step, and placing both offsprings in the neighborhood with given probability according to their fitness value. Using this model, he discovered that the cGA showed a fast convergence, although in a located way, that is why niches of individuals with fitness values close to the optima were formed in the population. This fast (and located) convergence is not surprising if we consider that the selection is really effective in very small populations. Therefore, we can conclude from this work that a characteristic behavior of cGAs is the forming of diverse niches in the population where the reproductive cycle tends to promote the specialization of the composing individuals (the exploitation inside these areas is promoted). From this statement we conclude that the cGA maintain diverse search paths towards different solutions, as each of these niches can be seen as an exploitation path of the search space.

In the same conference where Davidor presented the commented work, Spiessens and Manderick [232] published a comparative study of the temporal complexity between their cGA and a secuencial GA. Due to the problem dependance of the evaluation step, they ignored it in their studies, and they were able to demonstrate that the complexity of cGAs increases linearly according to the genotype length. On the contrary, the complexity of a secuencial GA increases polynomially according to the size of the population multiplied by the genotype length. As an increment in the length of the individual should theoretically be joined to an increase in the size of the population, an increment in the length of an individual will affect to the execution time of a sequential GA, but not in the case of the cGA. Moreover, in this article the authors deduce the expecting number of individuals when using the common selection methods in cGAs, showing that the proportional selection is the one with the lower selection pressure.

Now, we briefly discuss the results of the experiment presented some years later by Sarma and De Jong in [62]. In this work, they obtained a really important result for any researcher interested in developing cGAs. In their experiments, they compared some cGAs using diverse selection schemes, and they realized that two of the studied selections behaved in a different way even having equivalent selection pressures. In accordance with the authors, "*these results remark the importance of an analysis on the variation of selection schemes. Without this analysis, it is possible to fall into the trap of assuming that the selection algorithms which are expected to have an equivalent selection pressure produce a similar search behavior.*".

In 1994, Gordon et al. [110] studied seven cGAs with different neighborhoods on continuous and discrete optimization problems. In that paper, they concluded that larger neighborhoods work better with simple problems but on the contrary, with more complex problems it is better the use of smaller neighborhoods.

Capcarrère et al. defined in [44] a set of very useful statistical measures for understanding the dynamical behavior of cEAs. In that study two kinds of statistics were used: based on genotype and phenotype. The metrics based on genotype measure issues related to the chromosomes of the individuals of the population, whereas the ones based on phenotypes take account of the adequacy properties of the individual, basically in the fitness.

More recently, Alba et al. performed in [19] a comparative study of the behavior of cGAs with synchronous and asynchronous update policies of the population. The results obtained show that the asynchronous cGAs perform a higher selection pressure than the synchronous ones, so they converge faster, and generally, find the solution sooner than the synchronous in the less complex studied problems. On the contrary, in the case of the hardest problems, the synchronous cGAs seem to be the ones offering a better efficiency, as the asynchronous get stuck in local optima more frequently.

In [84], Eklund performed an empirical study for determining the most appropriate selection method and the shape and size of the neighborhood for a cellular GP. The conclusions were that both the ideal size and shape of the neighborhood depend on the size of the population. Regarding the selection method, any of the studied ones behave well with elitist populations. Moreover, it was discovered that the higher the number of dimensions in a population is, the higher the dispersion speed of the good solutions is and therefore the size of the population should be larger for obtaining a good behavior.

Finally, this book contributes to the theory in cEAs with new and severe theoretical and practical studies of the selection pressure in synchronous and asynchronous cGAs with different population shapes (Chap. 4) [17, 74], and comparative studies between cEAs and panmictic EAs (see Chap. 3 for a deep comparison in the field of GAs, and Chaps. 10 and 13 for the cases of EDAs [23] and memetic GAs [14], respectively).

2.4 Algorithmic Improvements to the Canonical Model

In this section we remark some other relevant publications in the field of cGAs which do not directly belong to any of the previous sections, but they deserve a mention because they suppose important advances in the field.

Rudolph and Sprave presented in [219] a synchronous cGA with structured population in a ring topology and with a self-adaptive acceptance threshold, which is used as a way to add elitism to the algorithm. The algorithm was compared to a panmictic GA, which resulted to have a considerably worse efficiency than the cGA.

In [111, 113], the authors presented an heterogeneous algorithm called Terrain-Based GA (TBGA). The idea of TBGA is that the programmer does not need to tune any of the parameters. This is achieved by defining a rank of values for each of the parameters, which disperse along the axis of the population of the cGA. So, in each position of the population there exists a different combination of the parameters, being similar the parameters of neighbor positions. The TBGA algorithm has been also used for finding a good parametrization for a cGA. The authors present two methods for searching a good configuration, and they lie in the storage of the number of times that the best individual of the population in every generation was in each position of the grid. The general idea is that the location with a higher number of best individuals along the different generations should have a good parametrization. In [113] Gordon and Thein conclude that the algorithm with the parameter configuration of that location with a higher registered number of best individuals has a really better efficiency than TBGA, and also than a manually tuned cGA.

In the literature, some cGAs hybridized with local search methods have been published. Some examples are the cGAs for training recurrence artificial neural networks for solving the long-term dependency problem [157] or the XOR function [156], and the most recently hybrid cGAs proposed for the SAT problem by Folino et al. [91, 92], and by Luo and Liu [171], where the mutation operator is replaced by a local search step. This last algorithm has the particularity of being developed for running in the Graphic Processing Unit (GPU) of the computer, instead of using the Central Processing Unit (CPU).

Some models have been also proposed where extinction of the individuals in particular areas of the population is introduced. For example, Kirley proposes CGAD [147, 149], a cGA with perturbations characterized by the possibility of disaster occurrences, which removes all the individuals located in a particular area of the population. CGAD was successfully tested in numerical, dynamical, and multi-objective [146] optimization problems, becoming the only multi-objective approach of a cellular model. Another similar proposal is the one presented by Krink et al. [155, 215], in which some disasters are frequently generated in areas of the population where the individuals are replaced by new individuals. The frequency of these disasters is controlled by a sand bag model (see [155] for more details).

Kirley also proposed, with Thomsen and Rickers, an EA based in religions (RBEA) in [244], where a set of religions is established in the population. In this model, individuals can only belong to one single religion, and they are allowed to mate only with individuals belonging to their same religion. This way, the creation of niches is promoted. Occasionally, individuals can become to another different religion. In this work, the authors demonstrated that the new RBEA improved the results of a panmictic EA and a cEA for a set of numerical functions.

Although cGAs have an implicit migration given by the overlap of the neighborhoods, some authors try to emphasize this issue by adding any other additional kind of explicit migration. Some examples of this class of algorithms are the ones presented in [112, 116], where the migration is introduced (by copying an individual anywhere in the population in determined periods of time) for allowing separate niches to interact. This kind of migration is also used in [160], where Lee et al. additionally propose another new migration policy consisting in applying the mutation operator to the migrating individuals. Another sort of migration is the one in the previously commented CGAD [147], where the extinguished areas are filled by replicas of the best individual. Finally, there is also an implicit migration in the model proposed by Alba and Troya in [26], explained in Sect. 2.2.2, as the change in the shape of the population implies a redistribution of part of the individuals composing it.

In [167] a new cGA was proposed, called *GALE*, for the data mining classification problem. The singularity of *GALE* with respect to a canonical cGA is that it allows the existence of empty cells in the grid. Therefore, the offsprings will be placed in the empty cells of their neighborhoods, and if there are not empty cells, they replace the worst individual in the neighborhood. Another singularity of this model is the existence of a *survival step*, where it is decided whether the individuals are kept for the next generation or not, in terms of the fitness values of each individual and its neighbors.

In 2002, Li and Sutherland presented in [165] a variation of cGA called prey/predator algorithm, where the preys (corresponding to the individuals representing potential solutions to the problem) move freely around the positions of the grid, mating with neighbor preys in each generation. Moreover, there exists a number of predators which are continuously displacing around the population, and they kill the weakest prey of their neighbor in each generation. The algorithm showed good results when comparing it to a panmictic GA and to a distributed heterogeneous GA (both from [127]) for a set of 4 numeric problems. This algorithm was later extended in [164] to the multi-objective domain with good results. In fact, there exist two more prey/predator algorithms proposed for multi-objective problems in the literature, published in [125] and [159].

Another sort of non standard cGA is the one given by the *co-evolutionary* approaches. The most well-known example is the Hillis method [129] for sorting minimal sorting networks. The Hillis proposal consisted in a massively parallel GA with two independent populations, which evolve according to an

standard cGA. In one population, the *hosts* represent sorting networks, meanwhile in the other population the *parasites* represent test cases. The fitness of the sorting networks is given by measuring how well they sort the test cases provided by the *parasites* at the same grid location. Conversely, the *parasites* are scored according to how well they find flaws in the corresponding sorting networks (in the same location of the grid of the *hosts*). In this way, the algorithm evolves for finding the solution to the problem, as in the population of the *parasites* the individuals evolve to more difficult test cases, meanwhile in the other population the sorting networks evolve to solve these more difficult study cases each time.

Another interesting co-evolutionary variation of the cGA model is the *cellular programming algorithm* by Sipper [230]. The cellular programming has been widely used for evolving cellular automata in order to make computational tasks, and it is based in the co-evolutionary topology of the cellular automata neighbor rules.

For ending this section, we briefly present the main existing works where cGAs have been applied on dynamical optimization problems. Some examples are the work of Kirley and Green [148], the previously mentioned CGAD [147] (also from Kirley), or the comparative study between the efficiency of the panmictic GAs (stationary state and generational) and their equivalent cellular model performed by Alba and Saucedo [24], from which we can conclude that, generally speaking, it is the best of the three algorithms (and also the stationary state panmictic GA in some cases).

Although we can find some proposals in the literature of cEAs applied to the multi-objective field, there only exist two orthodox models, namely cMOGA [18] and MOCell [195, 194]. In this book we present these two models of multi-objective cGA, which are adaptations of the cGA model, and we apply them for solving both a complex optimization problem from the telecommunications field, and also a wide set of problems from academical benchmarks. We also present in this work a new cGA model with hierarchial population [139], called HcGA, where the exploitation of the best solutions is promoted, maintaining the diversity in the population simultaneously.

2.5 Parallel Models of cEAs

As it has been previously commented, cEAs were initially developed in massively parallel machines, although there have also merged some other models more appropriate for the currently existing distributed architectures. In Table 2.2 it is shown a summary of the main existing parallel cEAs in the literature.

Some examples of cGAs developed on SIMD machines are those studied by Manderick and Spiessens [176] (later improved in [232]), Mühlenbein [185, 186], Gorges-Schleuter [115], Collins [50] and Davidor [58], where some individuals are located in a grid, restricting the selection and the recombination

Table 2.2. Brief summary of the main existing parallel cEAs

Algorithm	Reference	Model
Manderick & Spiessens	[176] (1989)	Parallel cGA on SIMD machines
ECO-GA	[58] (1991)	Neighborhood of 8 individuals. Two offsprings per step
HSDGA	[256] (1992)	Fine and coarse grained hierarchical GA
fgpGA	[35] (1993)	cGA with two individuals per processor
GAME	[235] (1993)	Generic library for constructing parallel models
PEGAsuS	[214] (1993)	Fine and coarse grained for MIMD
LICE	[233] (1994)	Cellular model of evolutionary strategy
RPL2	[239] (1994)	Fine and coarse grained; very flexible
Juille & Pollack	[141] (1996)	Cellular model of genetic programming
ASPARAGOS	[117] (1997)	Asynchronous. Local search applied if no improvement
dcGA	[53] (1998)	Cellular or steady state islands models
Gorges-Schleuter	[119] (1999)	Cellular model of evolutionary strategy
CAGE	[92] (2001)	Cellular model of genetic programming
MALLBA	[81] (2002)	Generic library for constructing parallel models in C++
Combined cGA	[192] (2003)	Population composed by some cellular sub-populations
ParadisEO	[42] (2004)	Generic library for constructing parallel models in C++
Weiner et al.	[257] (2004)	Cellular ES with a variable neighborhood structure
Meta-cGA	[172] (2005)	Parallel cGA for local area networks using MALLBA
PEGA	[76] (2007)	Island distributed cGA (for *grid computing*)

to small neighborhoods in the grid. ASPARAGOS, the model of Mühlenbein and Gorges-Schleuter, was implemented on a *transputers* network, with the population structured in a cyclic stair. Later it evolved including new structures and matching mechanisms [117] until it was constituted as an effective optimization tool [118].

We also would like to remark the works of Talbi and Bessière [242], where the use of small neighborhoods is studied, and the one by Baluja [35], where three models of cGAs and a GA distributed in islands are analyzed on a MasPar MP-1, obtaining as a result the best behavior of the cellular models. Though, Gordon and Whitley presented in [114] a study comparing a cGA to a coarse grained GA, being the results of the latter slightly better. In [152] a comparison between some cGAs and the equivalent sequential GA is presented, clearly showing the advantages of using the cellular model. We can find in [27] a more exhaustive comparison than the previous ones between a cGA, two panmictic GAs (steady state and generational GAs), and a GA distributed in an island model in terms of the temporal complexity, the selection pressure, the efficacy, and the efficiency, among others issues. The authors conclude the existence of an important superiority of the structured algorithms (cellular and island models) according to the non structured ones (the two panmictic GAs).

In 1993, Maruyama et al. proposed in [177] a version of a cGA on a system of machines in a local area network. In this algorithm, called DPGA, an individual is located in each processor and, in order to reduce the communication to the minimum, in each generation each processor sends a copy of its individual to another randomly chosen processor. In each processor there exists a list of *suspended* individuals, where the individuals are located when they arrive from other processors. When applying the genetic operators in

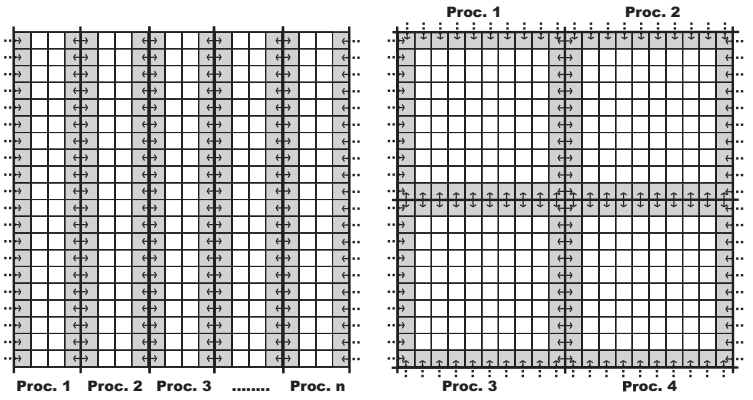


Fig. 2.1. CAGE (left) and the combined parallel model of cGA (right)

each processor, this list of *suspended* individuals behaves as the neighborhood. This model is compared to APGA, an asynchronous cGA proposed by Maruyama et al. [178], a sequential GA, and an specialized heuristic for the tackled problem. As a conclusion, the authors remark that DPGA shows a similar efficiency to the equivalent sequential algorithm.

There also exist more modern parallel cGA models, which work on connected computers in local area networks. These models should be designed for reducing the communications to the minimum as, due to their own characteristics, the cellular models need a high number of communications.

In this frame, Nakashima et al. propose in [191] a *combined cGA* where there exist some sub-populations with evolving cellular structure, and interacting through their borders. A graph of this model can be seen in the right part of Fig. 2.1. In a later work [192], the authors propose some parameterizations with different number of sub-populations, ways of replacement, and the topology of the sub-population, and they analyze the results. The authors used this model in a sequential machine, but it is directly extrapolated to a parallel model, where each processor contains one of the sub-populations.

Folino et al. propose in [93] CAGE, a parallel GP. In CAGE, the population is structured in a bidimensional toroidal grid, and it is divided in groups of columns which constitute sub-populations (see the graph on the left in Fig. 2.1). In this way, the number of messages to send is reduced according to other models which divide the population in two dimensions (axis x and y). In CAGE, each processor contains a determined number of columns which evolve, and at the end of the generation the two columns in the borders are sent to the neighbor processors, so that they can use these individuals as neighbors of the individuals located in the limits of its sub-population.

Finally, there exist some generic programming frameworks of parallel algorithms which offer eases for implementing any kind of parallel algorithm, including the considered cellular models. Some of these frameworks are GAME [235], ParadisEO [42] or Mallba [81].

To end this section, we remark the two parallel models of cGAs studied in this book. In Chap. 8 we present the meta-cGA, which was developed using the MALLBA framework, and PEGA, a new cellular GA distributed in islands which can be executed in local area network environments or in *computational grids*. PEGA was applied to the largest existing instances of the VRP problem, contributing to the state of the art with some new solutions.

2.6 Conclusions

In this chapter we explored most of the existing works in the field of cellular evolutionary algorithms. The analyzed issues include both the main publications in the field and the most recent trends which are currently emerging. This study allows us to acquire some (necessary) knowledge of the domain of cEAs to understand open research lines like hybridization with local search, multi-objective optimization, necessity of theoretical models, etc.

Part II

Characterizing Cellular Genetic Algorithms

3

On the Effects of Structuring the Population

I cannot imagine a God who rewards and punishes the objects of his creation and is but a reflection of human frailty.

Albert Einstein (1879 - 1955) – Physicist

Since this book is devoted to the study of cellular genetic algorithms, it makes sense to prove their efficiency, effectiveness, and efficacy by comparing them against other panmictic (well-spread tools) and decentralized genetic algorithms. The obvious goal of this such study is to show the reader we have something new that merits attention. In this line, the purpose of this chapter is to study the effects of structuring a population. With this study we will justify the suitability of cellular genetic algorithms for solving complex problems. We intend to reach that goal by comparing two different cGAs (differing only in the selection method of the first parent) versus two well known families of genetic algorithms with non-structured populations (a generational and a steady state GA). Additionally, we also compare the best one of the two proposed cGAs versus a different decentralized family of GAs: distributed GAs, in which the population is partitioned into several separate collaborating sub-populations.

The structure of this chapter is the following one. Section 3.1 describes two typical panmitic GAs that will be compared versus cGAs. The two studied cGAs, as well as a distributed GA, are presented in Sect. 3.2. All these algorithms are compared and analyzed later in Sect. 3.3. Finally, we give our main conclusions in Sect. 3.4.

3.1 Non-decentralized GAs

In this section, the two most well-known panmictic GAs are described. These two algorithms will be later experimentally compared versus the proposed cGAs. In a non-decentralized GA, there is not any structure in the population, so individuals can mate with any other individual in the population. Next two sections discuss on them.

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_3, © Springer Science+Business Media, LLC 2008

Algorithm 3.1 Pseudo-code of a canonical ssGA

```
1. proc Evolve(ssga)        // Parameters of the algorithm in 'ssga'
2. GenerateInitialPopulation(ssga.pop);
3. Evaluation(ssga.pop);
4. while ! StopCondition() do
5.    parents ← Selection(ssga.pop);
6.    offspring ← Recombination(ssga.Pc,parents);
7.    offspring ← Mutation(ssga.Pm,offspring);
8.    Evaluation(offspring);
9.    Replacement(ssga.pop,offspring);
10. end while
11. end proc Evolve
```

3.1.1 Steady State GA

A pseudo-code of the steady state GA (ssGA) is given in Alg. 3.1. As it can be seen, it is a (μ+1)-GA. In each generation, two parents are selected from the whole population with a given selection criterion (line 5). These two individuals are recombined (line 6) and then one of the obtained offsprings is mutated (line 7). The mutated individual is evaluated and then it is inserted back into the population, typically replacing the worst individual in the population (if the new one is better). This loop is repeated until the termination condition is met (line 4).

3.1.2 Generational GA

Generational GAs (genGAs) are a kind of non-structured methods in which any individual can interact with any other one in the population, as in the case of ssGAs. The difference between these two algorithmic families is that genGAs are (μ, λ)-GAs, so the newly generated individuals are placed in

Algorithm 3.2 Pseudo-code of a canonical genGA

```
1. proc Evolve(genga)        // Parameters of the algorithm in 'genga'
2. GenerateInitialPopulation(genga.pop);
3. Evaluation(genga.pop);
4. while ! StopCondition() do
5.    for i←0 to genga.popSize do
6.       parents ← Selection(genga.pop);
7.       offspring ← Recombination(genga.Pc,parents);
8.       offspring ← Mutation(genga.Pm,offspring);
9.       Evaluation(offspring);
10.      Add(auxiliary_pop,offspring);
11.   end for
12.   genga.pop ← auxiliary_pop;
13. end while
14. end proc Evolve
```

Algorithm 3.3 Pseudo-code of a canonical dGA

```
1. proc Evolve(dga)        // Parameters of the algorithm in 'dga'
2. for each island do in parallel:
3. {
4. GenerateInitialPopulation(pop);
5. Evaluation(pop);
6. while ! StopCondition() do
7.    parents ← Selection(pop);
8.    offspring ← Recombination(dga.Pc,parents);
9.    offspring ← Mutation(dga.Pm,offspring);
10.   Evaluation(offspring);
11.   Replacement(pop,offspring);
12.   if Migrate(dga.MigrFreq) then
13.      Migration(best_individual,neighbour_island);     // Synchronous case
14.      Replacement(Receive(individual),pop);
15.   end if
16. end while
17. }
18. end proc Evolve
```

an auxiliary population which will replace the current population when it is completely filled, i.e., when the number of newly generated solutions is equal to the size of this auxiliary population (see Alg. 3.2). In our case, the sizes of both the auxiliary and the current population is the same ($\mu = \lambda$).

3.2 Decentralized GAs

Decentralized GAs are characterized by their structured population. In a structured (or decentralized) population, individuals can only mate with a subset of the population instead of all the individuals.

The two main ways for structuring the population is to partition it into independent islands that share some information during the run or to establish an isolation by distance strategy among the individuals. They are distributed and cellular GAs, respectively.

Cellular GAs are described in Chap. 1. Thus, we only present in this section a pseudo-code of a typical distributed GA (dGA), which can be seen in Alg. 3.3. As it is shown, in this dGA the population is partitioned into a number of separate sub-populations in which independent ssGAs are executed (lines 4 to 11). Additionally, the islands exchange (possibly a copy of) their best individual with the nearest island (in a unidirectional ring topology) with a given migration frequency. When an island receives the best individual from another island, it replaces its worst individual with it. In this case, the received individual is not better than the worst one in the receiving population, no replacement is made (other options exist). Migration takes place at a given

Table 3.1. Parameterization used in the decentralized algorithms

Population size	400 individuals
Parent selection	Binary tournament + binary tournament
	Current individual + BT (for cGACenter+BT)
Recombination	DPX, $p_c = 1.0$
Mutation	Bit-flip, $p_m = 1.0/L$ (L = Individual length)
Replacement	Rep_if_not_Worse
Neighborhood	NEWS (only for the cGAs)
Lattice	20×20 (only for the cGAs)
Number of islands	5 (only for dGA)
Migration frequency	Every 10^4 local evaluations in each island (only for dGA)
Stop condition	Find the optimum or reach 10^6 evaluations

frequency determined by the number of isolated steps (other criteria are of course possible). In synchronous dGAs all populations do this step at the same time, thus proceeding in a sort of distributed synchronized steps. In asynchronous dGAs (more suited for heterogeneous hardware, Internet, and grid computing) the sending and the receiving steps are separated in the code, and an island can check for incoming individuals whenever appropriate regardless on the transmission of its own individuals.

We study in this chapter two different versions of cGAs and one dGA. In the next section, their parameterization is given, and all the algorithms (these three ones plus the two panmictic algorithms presented in Sect. 3.1) are numerically analyzed.

3.3 Experimental Comparison

The parameterization of the studied algorithms is described in Table 3.1. Specifically, we study two panmictic algorithms (genGA and ssGA), two cGAs (cGACenter+BT and cGABT+BT) and a dGA. The two cGAs differ each other only in the selection method for the first parent. In cGACenter+BT the first parent is the current individual itself, while in the case of cGABT+BT the first parent is selected by binary tournament, exactly as the other parent.

In all the studied algorithms, the population is composed of 400 individuals, from which parents are selected by binary tournament (BT), with the already mentioned exception of cGACenter+BT. The two parents are recombined applying the two points crossover operator (DPX) with probability 1.0 ($p_c = 1.0$). From the two obtained offsprings, we only consider the one having the largest portion of the best parent, and its alleles are mutated with equal probability by flipping their (binary) value. The resulting offspring replaces the selected individual in the population if it has a equal or better fitness value than the latter. The individual selected for the replacement is the worst one for the ssGA, the current one in case of the cellular GAs, and the worst individual in the island for dGA. In the case of genGA, a new auxiliary population that completely replaces the current one is built. All the algorithms were run in a single PC with a Pentium IV 2.8GHz processor under Linux operating system, and having 512MB of memory.

Specific parameters for the cGAs are the NEWS (or Von Neumann) neighborhood, and a lattice shape of 20×20 individuals. In the case of dGA, the population is partitioned into five smaller sub-populations (composed of 80 individuals each) that exchange their best individuals every 10^4 evaluations.

The algorithms are tested on a large benchmark composed of problems with many different features, such as epistasis, multimodality, problem generators, or parameter fitting. This guarantees a high level of confidence in the results, although the evaluation of conclusions will result more laborious than with a small test suite. Particularly, we solve the problems COUNTSAT, ECC, FMS, MAXCUT (three different instances), MMDP, MTTP (three different instances), P-PEAKS, and 3-SAT. The descriptions of these problems are given in Appendix A in order to make this book self-contained. Additionally, 100 independent runs were done of each algorithm for every problem, and statistical tests were applied to the results. These tests are applied in all the experiments along this book, and they lie in applying ANOVA (or Kruskal-Wallis) tests for normally (non-normally) distributed data. The application of these tests allows us to obtain concluding results with a 95% confidence level. The reader is referred to Chap. 5 for a deeper explanation on the statistical studies performed.

In Sect. 3.3.1 we compare the behavior of two cGAs versus two panmictic GAs, while these two same cGAs are compared versus the dGA in Sect. 3.3.2.

3.3.1 Cellular versus Panmictic GAs

The two panmictic GAs are compared versus the two proposed cGAs in terms of average evaluations to find a solution, time, and hit rate (percentage of successful runs) in Tables 3.2, 3.3, and 3.4, respectively. The standard deviation of the results is also provided. Symbol '+' in Tables 3.2, and 3.3 indicates that significant values were obtained in the comparison of the results provided by the algorithms with 95% confidence level (p-value bellow 0.05), while '−' means that no significant differences were found. The cases in which no comparisons could be made are marked with '•'. The best obtained results in these tables are **bolded**.

Table 3.2. Comparison of two cGAs versus two panmictic algorithms. Average (thousands) evaluations

Problem	ssGA	genGA	cGABT+BT	cGACenter+BT	Test
COUNTSAT	—	—	—	**3.20 ±170.90**	•
ECC	**24.41 ±490.28**	110.23 ±389.09	43.51 ±198.98	159.00 ± 28.06	+
FMS	—	392.88 ±161.60	**32.00 ± 0.00**	535.23 ±232.45	−
MAXCUT100	**29.05 ±191.24**	323.36 ±192.87	128.15 ±274.33	281.21 ±383.92	+
MAXCUT20_01	**2.60 ± 1.70**	6.35 ± 5.71	3.10 ± 0.98	4.73 ± 1.93	+
MAXCUT20_09	**4.66 ± 2.71**	13.53 ± 9.83	5.05 ± 2.08	7.79 ± 3.24	+
MMDP	—	596.37 ±219.40	—	**222.64 ±411.63**	+
MTTP20	**1.90 ± 0.47**	4.63 ± 1.35	3.12 ± 0.74	5.11 ± 1.21	+
MTTP100	**18.94 ± 6.85**	269.17 ±223.71	41.85 ± 6.43	164.40 ± 32.86	+
MTTP200	140.32 ± 172.10	611.83 ±222.66	**95.75 ± 21.41**	473.16 ± 68.39	+
P-PEAKS	**8.34 ± 0.83**	19.95 ± 1.82	17.96 ± 1.45	39.12 ± 3.10	+
SAT	—	230.11 ±393.04	**120.18 ±370.34**	202.71 ±129.45	+

Table 3.3. Comparison of two cGAs versus two panmictic algorithms. Average time (seconds)

Problem	ssGA	genGA	cGABT+BT	cGACenter+BT	Test
COUNTSAT	—	—	—	**0.10** ± **0.12**	•
ECC	93.73 ± 14.63	10.36 ±12.99	**6.66** ± **1.34**	12.62 ± 2.16	+
FMS	—	68.34 ±55.29	**4.91** ± **0.00**	78.83 ±34.25	−
MAXCUT100	81.57 ± 19.65	32.21 ±33.84	**21.78** ±**30.22**	24.58 ±17.04	+
MAXCUT20_01	6.77 ± 3.01	0.21 ± 0.15	0.22 ± 0.05	**0.17** ± **0.05**	+
MAXCUT20_09	10.35 ± 3.09	0.38 ± 0.24	0.31 ± 0.10	**0.25** ± **0.07**	+
MMDP	—	59.46 ±19.58	—	**16.26** ±**13.73**	+
MTTP20	5.14 ± 1.44	**0.15** ± **0.04**	0.21 ± 0.05	0.17 ± 0.03	+
MTTP100	62.63 ± 8.13	12.09 ± 8.94	**2.79** ± **0.77**	6.56 ± 1.25	+
MTTP200	308.91 ±189.91	45.15 ±16.51	**9.87** ± **2.35**	31.04 ± 4.63	+
P-PEAKS	41.14 ± 5.29	**4.67** ± **0.43**	6.27 ± 1.83	8.70 ± 0.68	+
SAT	—	25.56 ±23.80	**12.33** ±**17.64**	20.73 ±12.94	+

Table 3.4. Comparison of two cGAs versus two panmictic algorithms. Hit rate

Problem	ssGA	genGA	cGABT+BT	cGACenter+BT
COUNTSAT	0.0%	0.0%	0.0%	**3.0**%
ECC	50.0%	78.0%	96.0%	**100.0**%
FMS	0.0%	6.0%	1.0%	**27.0**%
MAXCUT100	4.0%	7.0%	11.0%	**45.0**%
MAXCUT20_01	**100.0**%	**100.0**%	**100.0**%	**100.0**%
MAXCUT20_09	**100.0**%	**100.0**%	**100.0**%	**100.0**%
MMDP	0.0%	**90.0**%	0.0%	47.0%
MTTP20	**100.0**%	**100.0**%	**100.0**%	**100.0**%
MTTP100	**100.0**%	98.0%	**100.0**%	**100.0**%
MTTP200	99.0%	22.0%	**100.0**%	**100.0**%
P-PEAKS	**100.0**%	**100.0**%	**100.0**%	**100.0**%
SAT	0.0%	72.0%	83.0%	**100.0**%

As it can be seen in Table 3.2, ssGA is the most efficient of the four compared algorithms since it performs the lowest number of evaluations (in average) for solving 7 out of the 12 studied problems. In contrast, ssGA could not solve the most difficult problems in any of the 100 independent runs made (see Table 3.4). Moreover, ssGA shows a very erratic behavior, since the standard deviations obtained for ECC, MAXCUT100, and MTTP200 problems are very high. The other studied panmictic GA, genGA, is clearly the worst of the four algorithms in terms of efficiency. The differences of genGA with respect to the most efficient algorithm for each problem have always statistical confidence (with the exception of FMS).

Regarding the two compared cGAs, it can be seen that in general they are less efficient than the ssGA, but in contrast their behavior is more robust, since they obtain smaller standard deviations (in proportion to the average number of evaluations) than the panmictic algorithms. Additionally, it stands out the most exploitative behavior of cGABT+BT with respect to cGACenter+BT, since it finds a solution faster. As a consequence, cGABT+BT can more easily get stuck into local optima when facing some the more complex problems.

If we now pay attention to the elapsed time, we find opposite results. In this case, the best (faster) algorithms are the two cGAs, followed by genGA, and finally ssGA. Although ssGA is the algorithm that performs the lowest number of fitness function evaluations in average for most of the problems (Table 3.2), it is the slowest algorithm in all the problems (Table 3.3). The reason for this

fact is that the algorithm needs to look for the best and worst individuals after each insertion step of an offspring into the population. Comparing the two cGAs one each other, cGABT+BT is faster (differences are significant in all cases), although it cannot find the solution for problems COUNTSAT and MMPD.

In terms of the percentage of successful runs (hit rate), it can be seen in Table 3.4 that cGACenter+BT is the best algorithm in all the problems, except for MMDP. Additionally, it is the only algorithm able to find the best-known solution for all the studied problems. In this sense, it stands out the case of COUNTSAT, for which cGACenter+BT is the unique studied algorithm that finds the optimal solution.

In summary, ssGA is the most exploitative algorithm of the four compared ones. This allows it to find the best solution numerically faster with respect to the other compared algorithms, but when facing hard problems it rapidly falls into local optimal solutions from which it cannot scape. In contrast, the two proposed cGAs perform a better tradeoff between exploration and exploitation. This allows the algorithms to scape from local optima, but penalizing the convergence speed. Indeed, these two algorithms (and particularly cGACenter+BT) are the best ones in terms of efficacy (hit rate). Moreover, an additional advantage of the cGAs is that their structured population allows them to evolve in a shorter time than the panmictic GAs, since they are faster than the other two panmictic GAs despite the number of evaluations they perform is usually higher.

3.3.2 Cellular versus Distributed GAs

After demonstrating in Sect. 3.3.1 that the two proposed cGAs perform a better exploration/exploitation tradeoff in the search space than the panmictic GAs we compare now in this section the same two cGAs versus another decentralized GA: the distributed GA. The results are shown in terms of efficiency, time, and efficacy in Tables 3.5, 3.6, and 3.7, respectively. As in Sect. 3.3.1, values in Tables 3.5 and 3.6 are the average over 100 runs plus their standard deviation. Table 3.7 presents the percentage of runs in which the algorithm found the best-known solution for each problem.

We can see in Table 3.5 that there is not a clear best algorithm in terms of efficiency among the studied structured GAs. Perhaps, dGA can be considered to be better than the two compared cGAs because it obtained the best results for 5 out of the 9 problems it can solve, but cGABT+BT is the best one in the other 4 problems, plus FMS (not solved at all by dGA). Differences between dGA and cGABT+BT are significant except for problems MAXCUT100 and SAT. The high exploration capabilities of cGACenter+BT are clear, although it performs the highest number of evaluations for all the problems it is the only algorithm solving all the tackled problems (COUNTSAT and MMDP are not solved by the other two algorithms).

Table 3.5. Comparison of the decentralized GAs. Average (thousands) evaluations

Problem	dGA	cGABT+BT	cGACenter+BT	Test
COUNTSAT	—	—	**3.20 ±170.90**	•
ECC	**36.36 ±270.54**	43.51 ± 198.98	159.00 ± 28.06	+
FMS	—	**32.00 ± 0.00**	535.23 ± 232.45	•
MAXCUT100	**85.71 ±277.55**	128.15 ± 274.33	281.21 ± 383.92	+
MAXCUT20_01	**2.23 ± 0.75**	3.10 ± 0.98	4.73 ± 1.93	+
MAXCUT20_09	6.16 ± 22.08	**5.05 ± 2.08**	7.79 ± 3.24	+
MMDP	—	—	**222.64 ±411.63**	•
MTTP20	**2.02 ± 0.42**	3.12 ± 0.74	5.11 ± 1.21	+
MTTP100	93.98 ± 75.30	**41.85 ± 6.43**	164.40 ± 32.86	+
MTTP200	623.11 ± 215.05	**95.75 ± 21.41**	473.16 ± 68.39	+
P-PEAKS	**9.98 ± 0.74**	17.96 ± 1.45	39.12 ± 3.10	+
SAT	122.24 ± 191.62	**120.18 ±370.34**	202.71 ± 129.45	+

Table 3.6. Comparison of the decentralized GAs. Average time (seconds)

Problem	dGA	cGABT+BT	cGACenter+BT	Test
COUNTSAT	—	—	**0.10 ± 0.12**	•
ECC	15.92 ±16.31	**6.66 ± 1.34**	12.62 ± 2.16	+
FMS	—	**4.91 ± 0.00**	78.83 ±34.25	•
MAXCUT100	33.61 ±33.15	**21.78 ±30.22**	24.58 ±17.04	−
MAXCUT20_01	1.24 ± 0.41	0.22 ± 0.05	**0.17 ± 0.05**	+
MAXCUT20_09	1.98 ± 2.34	0.31 ± 0.10	**0.25 ± 0.07**	+
MMDP	—	—	**16.26 ±13.73**	•
MTTP20	1.14 ± 0.23	0.21 ± 0.05	**0.17 ± 0.03**	+
MTTP100	28.66 ±16.49	**2.79 ± 0.77**	6.56 ± 1.25	+
MTTP200	203.60 ±64.12	**9.87 ± 2.35**	31.04 ± 4.63	+
P-PEAKS	9.07 ± 0.62	**6.27 ± 1.83**	8.70 ± 0.68	+
SAT	79.15 ±77.95	**12.33 ±17.64**	20.73 ±12.94	+

Table 3.7. Comparison of the decentralized GAs. Hit rate

Problem	dGA	cGABT+BT	cGACenter+BT
COUNTSAT	0.0%	0.0%	**3.0%**
ECC	92.0%	96.0%	**100.0%**
FMS	0.0%	1.0%	**28.0%**
MAXCUT100	10.0%	11.0%	**45.0%**
MAXCUT20_01	**100.0%**	**100.0%**	**100.0%**
MAXCUT20_09	**100.0%**	**100.0%**	**100.0%**
MMDP	0.0%	0.0%	**47.0%**
MTTP20	**100.0%**	**100.0%**	**100.0%**
MTTP100	**100.0%**	**100.0%**	**100.0%**
MTTP200	31.0%	**100.0%**	**100.0%**
P-PEAKS	**100.0%**	**100.0%**	**100.0%**
SAT	98.0%	83.0%	**100.0%**

As it happened in Sect. 3.3.1, the two cGAs are faster than the compared algorithm in this case too (as shown in Table 3.6). The reason is probably because a ssGA (the slowest algorithm of all the compared ones) is running inside each island of the dGA (with a population of 80 individuals). Additionally, the convergence of the population of every island is faster than in the panmictic ssGA presented in Sect. 3.3.1 due to its smaller size, diminishing the computational cost and thus the run time.

The cGACenter+BT algorithm is the one reporting the highest success rate (see Table 3.7), as it happened in the comparison versus the panmictic algorithms. Although the distributed GA performs in general a better exploration/exploitation tradeoff than the two panmictic algorithms studied in Sect. 3.3.1 (higher success rate) it seems that again the cellular algorithms are still better on an overall evaluation.

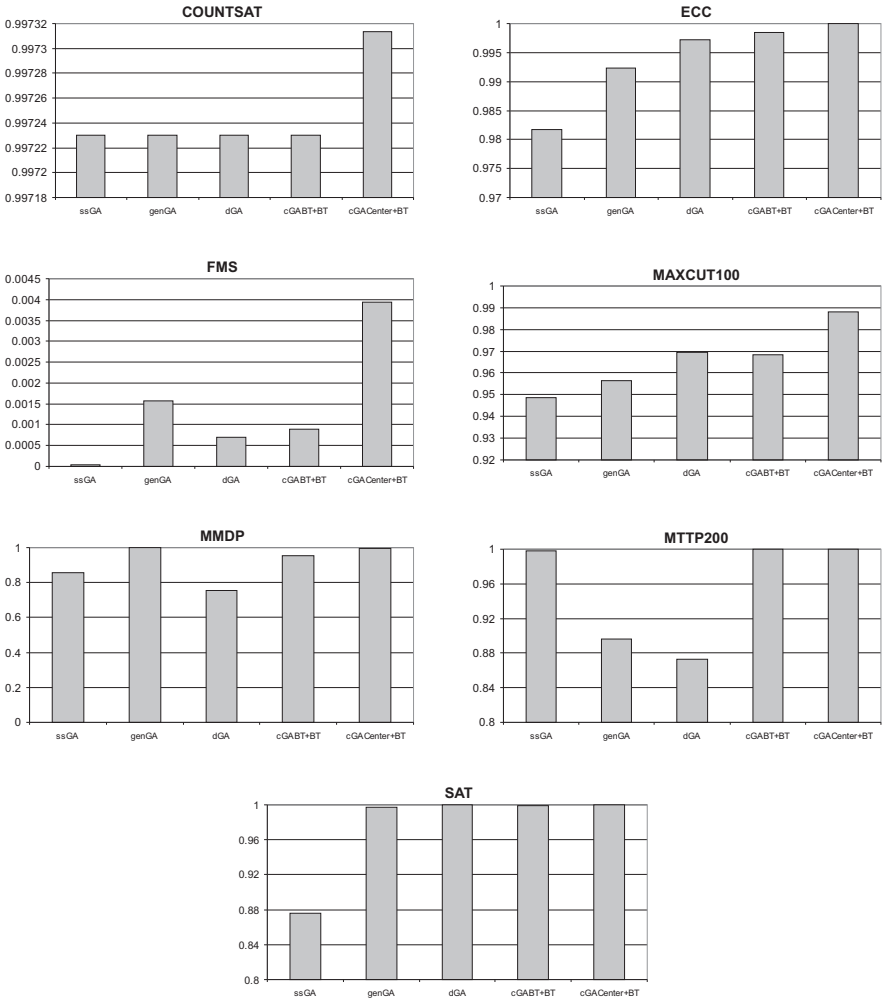


Fig. 3.1. Comparison of the average fitness (accuracy) found by all the algorithms for every problem

Finally, we end this section by comparing all the studied algorithms in terms of their accuracy. For that, we show in Fig. 3.1 the average fitness solution found by all the algorithms in the 100 runs made for every problem. The problems for which all the algorithms obtained a 100% hit rate are not plotted (MAXCUT20_01, MAXCUT20_09, MTTP20, MTTP100, and P-PEAKS). As it can be seen, the main conclusion we can draw from Fig. 3.1 is that cGA-Center+BT is the most accurate algorithm for all the problems, while the two panmictic algorithms (ssGA and genGA) are in general the less accurate ones. Algorithm cGABT+BT usually obtains better results than dGA, and these two algorithms report intermediate results: they are better than the panmictic ones but worse than the overall best cGACenter+BT.

3.4 Conclusions

We compared in this chapter the behavior of two different cGAs versus two panmictic GAs and a dGA in order to justify the good exploration/exploitation tradeoff usually performed by cGAs. We obtained that ssGA (a panmictic GA) is the most exploitative algorithm, and thus when it finds the solution to a problem it is the algorithm that requires the least number of fitness function evaluations. However, a consequence of this highly exploitative behavior is a low hit rate (robustness), since it easily gets stuck in local optima. This is specially undesirable for many problems for which it is then unable to find the optimal solution at all. Moreover, due to the required operations for managing the population of ssGA, this algorithm is the slowest among the compared ones. The other studied panmictic GA, namely genGA, is markedly the worst of all the compared GAs in any sense (although it is still very popular for practitioners).

The studied decentralized GAs perform a better exploration/exploitation tradeoff than the non-decentralized ones. They report, in general, better hit rates and execution times than the panmictic algorithms, although the number of evaluations performed is higher. Thus, they provide a better explorative behavior but penalizing their exploitation capabilities. This looks bad for academic benchmarks, but it is very important in real and complex tasks, where this smooth search leads to an actually useful solution. Comparing the studied decentralized algorithms among them, we conclude that the two cGAs are better than dGA in terms of both hit rate and run time, and there is not a clear better algorithm regarding the average number of evaluations needed to find the solution. Maybe the cGACenter+BT could be said to have the best behavior for our diverse testbed.

4

Some Theory: A Selection Pressure Study on cGAs

Do not worry about your difficulties in Mathematics. I can assure you mine are still greater.

Albert Einstein (1879 - 1955) – Physicist

This chapter addresses an important issue in any new methodological advance: the theoretical background. It is true that the formalization of metaheuristics in general has always been a minor component in the international arena. This situation has greatly improved during the last years, with mathematical tools as Markov chains, Fourier analysis, theory of formae, no free lunch, and selection pressure theories, to name a few [33, 211]. We explicitly address here a tiny portion of mathematics characterizing the behavior of cellular EAs/GAs.

In order to better understand the functioning of cellular genetic algorithms, it is useful to characterize their selection pressure, a basic property that highly influences the behavior of the algorithm. As it was said before in previous chapters, any algorithm needs an adequate balance between exploration and exploitation to be useful in practice. The selection pressure in population based algorithms is a key issue that represents roughly this balance. Thus, this chapter is devoted to present and analyze the main existing works for mathematically characterizing the selection pressure of cGAs.

The variations in both the shape of the topology and the neighborhood definition can lead to many different algorithms, but a methodological study is possible by defining a *radius* for the population shape and the neighborhood (defined in Sect. 1.5.2). The relationship between them, called *ratio* (see Sect. 1.5.2), defines the characteristics of the search, as it influences the exploration/exploitation balance that the algorithm applies on the population of solutions. There exist a very few papers trying to mathematically characterize the behavior of cGAs in the literature, and only some of them propose equations for modelling the behavior of the selection pressure of the algorithm in terms of the different possible neighborhood and/or population shapes.

This chapter begins with a definition of the selection pressure and the *takeover time* (in Sect. 4.1). Then, in Sect. 4.2 we make a theoretical study on the behavior of cEAs according to the ratio. In this study we present and analyze the existing mathematical models in the literature for approaching the selection pressure curve of cEAs. Finally, we compare in Sect. 4.3 the

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_4, © Springer Science+Business Media, LLC 2008

theoretical results obtained by studying the behavior of some cGAs with different ratios on a set of seven kinds of complex problems (belonging to the combinatorial and numerical optimization fields) to sustain our conclusions. We will discuss on how difficult is to decide what is the best ratio according to the set of problems.

4.1 The Selection Pressure

The *selection pressure* is related to the concept of the *takeover time*, which is defined as the necessary time for a single (best) individual to colonize the whole population with copies of itself by using only the selection operator [106, 218]. Shorter *takeover* times indicate a higher intensity in the selection applied by the algorithm on the population (a higher selection pressure), and therefore, a higher convergence speed of the algorithm.

Algorithms with a similar *ratio* value show a similar selection pressure, as it is concluded in [221]. In Fig. 4.1 we show the similar behavior of two cellular algorithms with different neighborhood and population radii, but with similar ratio values. The algorithms shown in the graph use a NEWS neighborhood with a population of 32×32 individuals, and a C21 neighborhood with a population of 64×64 individuals. The ordinate axis represents the proportion of copies of the best individual in the population along time.

Therefore, it is very interesting to study how some parameters of the cEA influence the behavior on the search of the algorithm. Specifying, the individuals update policy in the population is a very influential parameter in the selection pressure of the algorithm, which has been theoretically studied in previous works [99, 102]. Thus, we can observe different behaviors of the algorithm by maintaining the shape of the neighborhood and the grid constant (i.e., L5 and square population) but using different individuals update policies. Indeed, we can observe in Fig. 4.2 that the global selection pressure induced by the many asynchronous policies is in the middle between the limits of the synchronous case and the high selection pressure of the panmictic case.

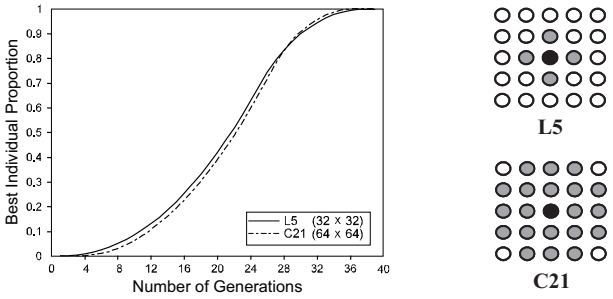


Fig. 4.1. Growth curves of the best individual proportion for two cEAs with different neighborhood and population shapes, but with similar resulting ratio values

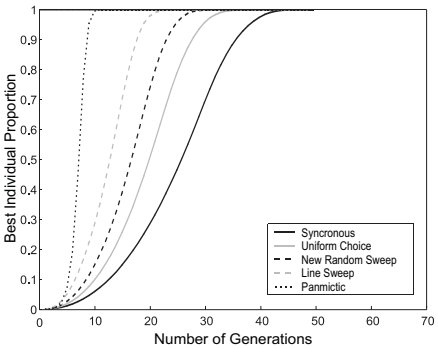


Fig. 4.2. Selection pressure of several identical cGAs, having different update policies (binary tournament selection)

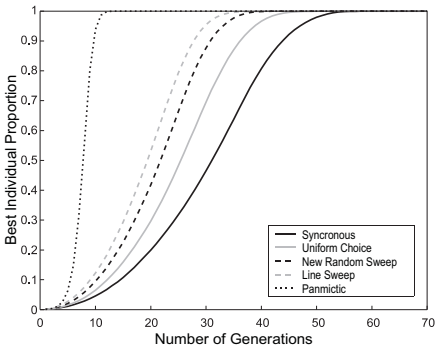


Fig. 4.3. Selection pressure of several identical cGAs, having different update policies (roulette wheel selection)

Therefore, we can influence the exploration or exploitation capacities of the search of the algorithm by changing the individuals update policy. The graph shown in that figure was obtained by computing the average of 100 independent runs, using a binary tournament selection on a population of 32×32 individuals with a L5 neighborhood. The binary tournament selection consists of randomly selecting two individuals from the neighborhood and choosing the best of them.

Another parameter that influences the selection pressure of cGAs is the parent selection operator. The cGAs shown in the selection pressure graphs in Fig. 4.2 use binary tournament selection. We can observe the differences with the curves shown in Fig. 4.3, which are the selection pressure curves of the same algorithms but using the roulette wheel selection operator. As we can see in these two figures, the roulette wheel selection induces a lower selection pressure (longer *takeover* times) in the algorithm than binary tournament.

Finally, it is also interesting to study the influence of the ratio between the neighborhood and population radii on the selection pressure of a cEA [17, 74]. For that, we calculate the selection pressure graphs for some cEAs with the only difference of the population shape used, while keeping constant its size. We must remark that, as there are no operators included, the presented results for cGAs have a wide extension to other cEAs (we repeat this important fact several times in order to remember it). In Fig. 4.4 it is shown the selection pressure of different synchronous cEAs using L5 neighborhood and 6 possible grid shapes for a population of 1024 individuals. As we did with asynchronous cEAs, we obtain the results as the average of 100 independent runs of each algorithm. As it can be seen, the selection pressure on rectangular grids in synchronous cEAs is below the one of the synchronous cEA curve using the square grid (population of 32×32 individuals), which means that narrower grids motivate an explorative search. Notice that in Fig. 4.4, the x-axis differences are shown in logarithmic scale.

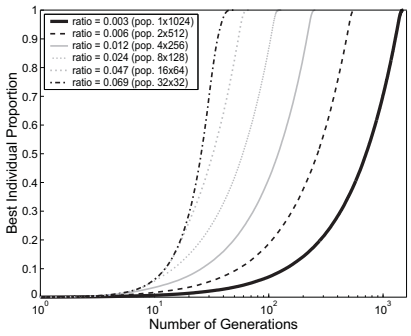


Fig. 4.4. Selection pressure of identical cGAs having different population shapes (binary tournament selection)

4.2 Theoretical Study

Apart from the *takeover* time, the growth curves are another very important issue for analyzing the behavior of an EA. The behavior of distributed EAs are characterized in [21] in terms of *takeover* time and the growth curves according to the percentage and the migration frequency of the individuals among the islands.

As we showed in Chap. 2, many studies attempt to characterize mathematically the behavior of cellular EAs, but only two of them consider the possibility of using rectangular populations [72, 102], although the ratio between the neighborhood and the population markedly influences the behavior of a cEA (see Fig. 4.4). In this section, we study how do the existing models approximate the behavior of different cGAs with 25 ratio values.

4.2.1 Approach to the Deterministic Model

If a deterministic behavior in the selection pressure of a cEA is considered, the best neighbor of each individual is always selected, therefore, if an individual has a neighbor with high fitness, in the next generation it will be in the set of best individuals. Consequently, only the high fitness individuals expand through the cells situated in the limit of the best individuals area, with a probability of 100%.

When using a rectangular population, the proportion of best individuals in the population grows quadratically until the limits of the population are reached (on the narrowest side), as it can be seen in Fig. 4.5a. Then, the growth of the proportion of best individuals in the population from the first two borders (narrowest part) until the other borders (widest part) are reached is linear (see Fig. 4.5b). Finally, the curve decreases after that in the same proportion as it grew before in the first part of the growth (Fig. 4.5c). In Fig. 4.6 it is shown a selection pressure of a cGA with a rectangular population (of size $M \times N$ with $M \leq N$) and deterministic growth. It can be clearly

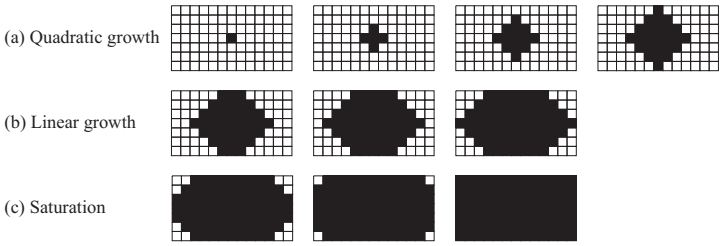


Fig. 4.5. Deterministic growth in a cEA with a rectangular population

seen that the growth of the number of best individuals in the population is quadratic until the moment $t = M/2$ (when the best individual spreads out to the borders of the population in the narrowest part, as seen in Fig. 4.5a), then the number of best individuals grows in a lineal way until the other borders of the population are reached (in the wide part, as it can be seen in Fig. 4.5b), in $t = N/2$ time. Finally, in the third step (Fig. 4.5c), the curve decreases as fast as it grew in the first step. In Eq. 4.1 this growth curve is mathematically defined in terms of two parameters (a and b).

$$N(t) = \begin{cases} a\,t^2 & \text{if } t < M/2 \ , \\ b\,t & \text{if } t < N/2 \ , \\ -a\,(T-t)^2 + 1 & \text{in other case} \ . \end{cases} \tag{4.1}$$

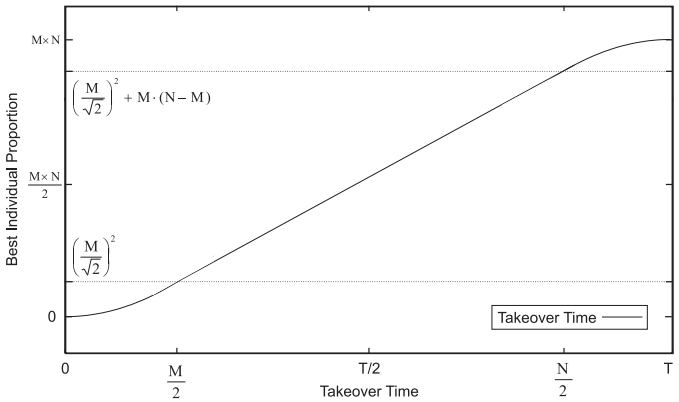


Fig. 4.6. Growth of the best individual in a cGA with M×N (M≤N) population, assuming a deterministic behavior in the selection pressure

4.2.2 A Probabilistic Model for Approaching the Selection Pressure Curve

In this section we present the best existing approach in the literature for mathematically fitting the selection pressure curve of cGAs with different ratio values; this approach is called the probabilistic model, and it was originally proposed in [72]. After that, it will be compared versus the main other existing proposals in the literature in Sect. 4.2.3.

The probabilistic model is a non parameterized mathematical approach that accurately fits the selection pressure curve of real cGAs with different selection operators and different ratio values. This mathematical model is based on probabilities, as it can be seen in Eq. 4.2.

As it was proposed in Sect. 4.2.1, it is considered in this model that the selection pressure curve of a cGA can be divided into three parts. This allows us to approximate our function in the case of both rectangular and square population shapes. In the case of a real model, it should be taken into account that the use of distinct selection methods implies a different spread of the number of best individuals through the population. Consequently, the diffusion speed of the best solutions along the population depends on the selection method used, and it is lower than in the deterministic case. Therefore, the transition limits between the different functions of the model that were fixed in the deterministic case must be weighted by a dependent factor of the selection method used. This factor is called s and it is a parameter which represents the expansion speed of the best individuals along the population. The probabilistic model is presented in Eq. 4.2, where $T(0) = 1$ is not specified for the sake of simplicity. Values of p_1 and p_2 are the probabilities that a given low fitness individual having one or two best individuals among its neighbors, respectively, becomes a new best individual in the next generation. The values of these two probabilities depend on the selection scheme used, and they are defined for three typical selection schemes later in this section. The parameter s is defined as the probability of a low fitness individual placed on the border of the set of best individuals becomes part of this set in the next generation (i.e., $s = p_2$).

The first function approximates the quadratic growth of the curve from $t = 0$ to $t = 1/s \cdot M/2$ (assuming $M \leq N$), when the borders of the expansion of best individuals in the population are connected (through the shortest path). This first equation computes the sum of the high fitness individuals in the previous time step $T(t-1)$ plus the individuals that will become part of the best ones for the next generation. Assuming that the region of best individuals grows in a square shape manner in this first phase (as shown in Fig. 4.5a), a low fitness individual becomes a high fitness one with different probability in terms of its location. If it is close to the sides of this square it will have up to two high fitness neighbors. Additionally, there are four individuals close to the corners of the square that will have a maximum of only one high fitness neighbor.

From the point $t = 1/s \cdot M/2$ to the moment when the set of the best individuals are connected through the largest path $t = 1/s \cdot N/2$, the number of low fitness individuals that becomes high fitness individuals in each generation follows a lineal growth. As it can be seen in Fig. 4.5b, in this case $2 \cdot (M-1)$ low fitness individuals will have up to two high fitness neighbors, and only two individuals (those in the two corners of the area of best individuals) will have a maximum of 1 high fitness neighbor.

Finally, the third part of this approach is a saturation function ending when no low fitness individuals are left in the population.

$$T(t) = \begin{cases} T(t-1) + 4 \cdot (t-1) \cdot (p_2 \cdot s^2 + 2 \cdot (1-s) \cdot p_1 \cdot s) + 4 \cdot p_1 \cdot s & t < \frac{M}{2} \cdot \frac{1}{s} \ , \\ T(t-1) + 2 \cdot (M-1) \cdot p_2 + 2 \cdot p_1 \cdot s & \frac{M}{2} \cdot \frac{1}{s} \leq t < \frac{N}{2} \cdot \frac{1}{s} \ , \\ T(t-1) + 4 \cdot \sum_{j=0}^{\lfloor \frac{M-1}{2} \rfloor - 1} \left(\left\lfloor \frac{M-1}{2} \right\rfloor - j \right) \cdot (p_2 + 4 \cdot (1-s) \cdot p_1 \cdot s) & \text{otherwise} \ . \end{cases} \tag{4.2}$$

In the rest of this section we study the accuracy of the probabilistic model for 75 different cGAs, and define the values of the probabilities used in Eq. 4.2. As it can be seen in Table 4.1, these cGAs are composed by a population of near 4096 individuals, and 25 different grid shapes were studied. The neighborhood used is NEWS in all the cGAs. Notice that the use of different grid shapes with a constant neighborhood for comparing cGAs with different ratio values does not restrict our results, as it was shown in Chap. 1. One of the parents is always the current individual, while the other one can be chosen by roulette wheel, binary tournament, or linear ranking selections. As it was said before, in the selection pressure study of an algorithm no recombination or mutation operators are used and the current individual is replaced by the best of its parents if the latter is better than the former. In this work, the error between the selection pressure curve approached by the probabilistic model and that of the real cGA is calculated for the 75 studied configurations. Additionally, some figures are given in order to graphically show how does this approach fit the true behavior of the real algorithm.

Table 4.1. Parameterization used in the cGAs

Population size	≈4096 individuals
Recombination	None
Mutation	None
Replacement	Repl_if_Better
Best individuals fitness	1.0
Worst individuals fitness	0.0

The error is calculated as the average of all the different studied populations of the mean square error between the approaching model and the experimental results. That is, let **a** and **b** be the obtained values of the probabilistic approach and the cGA, respectively, then the error Δ between the approach and the cGA for the 25 studied populations is defined as:

$$\Delta = \frac{\sum_{i=1}^{25} \text{MSE}(\text{model}_i)}{25} \;\; ; \;\; \text{MSE}(\text{model}) = \frac{1}{l} \sum_{i=0}^{l} (\mathbf{a}[i] - \mathbf{b}[i])^2 \;\; , \qquad (4.3)$$

being l the length of vectors **a** and **b**.

Roulette Wheel Selection

The roulette wheel selection lies in selecting each parent, among all the individuals of the neighborhood, with a given probability that is defined as the relationship between its fitness value and the sum of the fitness values of all the individuals in the neighborhood. Mathematically, the probability of choosing an individual i as a parent when using roulette wheel selection is described in Eq. 4.4.

$$p_i = \frac{\text{fitness}(i)}{\sum_{j \in \text{Neighborhood}} \text{fitness}(j)} \;\; . \qquad (4.4)$$

When studying the selection pressure, there only exist two possible fitness values for an individual. These values are 1.0 and 0.0. Therefore, according to Eq. 4.4, the probability of choosing a low fitness individual (with fitness = 0.0) by the roulette wheel method is 0.0, so if any high fitness neighbor exists (with fitness = 1.0) in the neighborhood of the considered individual, it will be always selected (or any other with the same fitness value) as a parent. Therefore, both p_1 and p_2 have value 1.0 in this model, and consequently when using roulette wheel selection the algorithm shows a deterministic behavior.

We show in Fig. 4.7 the obtained approach to the selection pressure curves of four cGAs with different population shapes by the proposed probabilistic function (notice the different scales on the number of generations in the graphs). As we can see, the adjustment is highly accurate in all the four cases, being the mean error obtained for all the 25 studied populations $\Delta = 1.4904 \cdot 10^{-6}$.

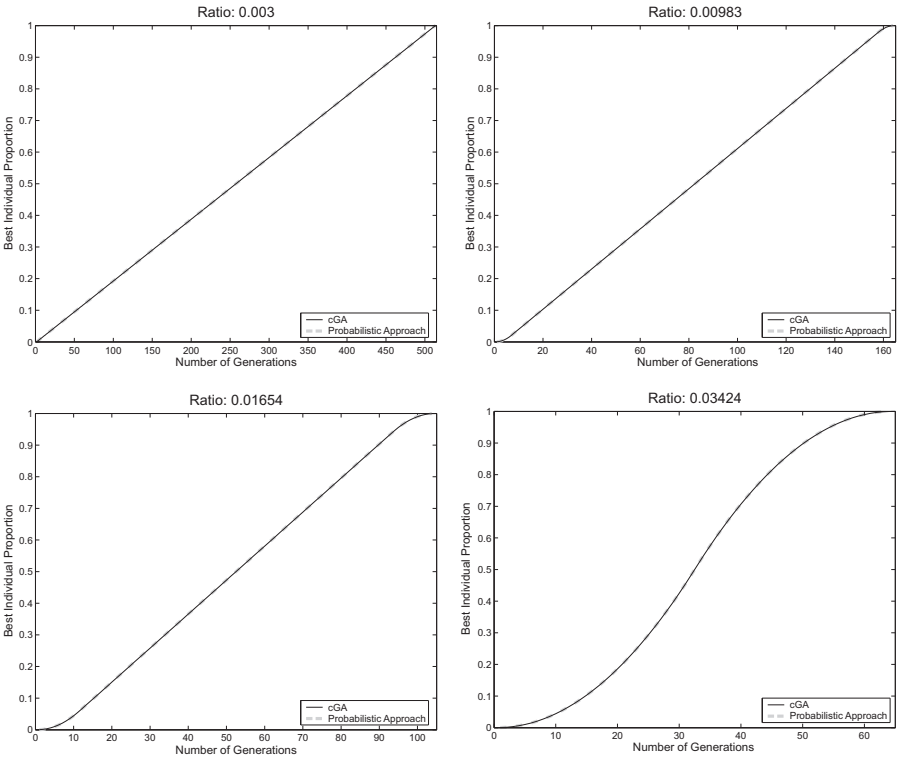


Fig. 4.7. Probabilistic approach to the convergence curves of four cGAs (roulette wheel selection)

Binary Tournament Selection

In the binary tournament selection, two individuals are randomly chosen from the neighborhood, and the best one is considered as parent. Therefore, if there exist two individuals in the neighborhood with fitness = 1.0, the probability of choosing one of them as parent is $p_2 = 16/25$, meanwhile, if there only exists one best individual in the neighborhood, the probability of selecting it is $p_1 = 9/25$.

In Fig. 4.8, we can observe the curves obtained by the probabilistic approach and the cGA for the same four populations studied in Fig. 4.7. As we can see, although the approach is not as good as in the roulette wheel case, we consider it is a good approach because the error is $\Delta = 2.5788 \cdot 10^{-4}$, lower than that of the other existing models in the literature, as we will see in Sect. 4.2.3. Moreover, we should take into account that the approach in this case is more complicated than using roulette wheel selection, since the behavior of the cGA is not deterministic, that is why the error loses two orders of magnitude.

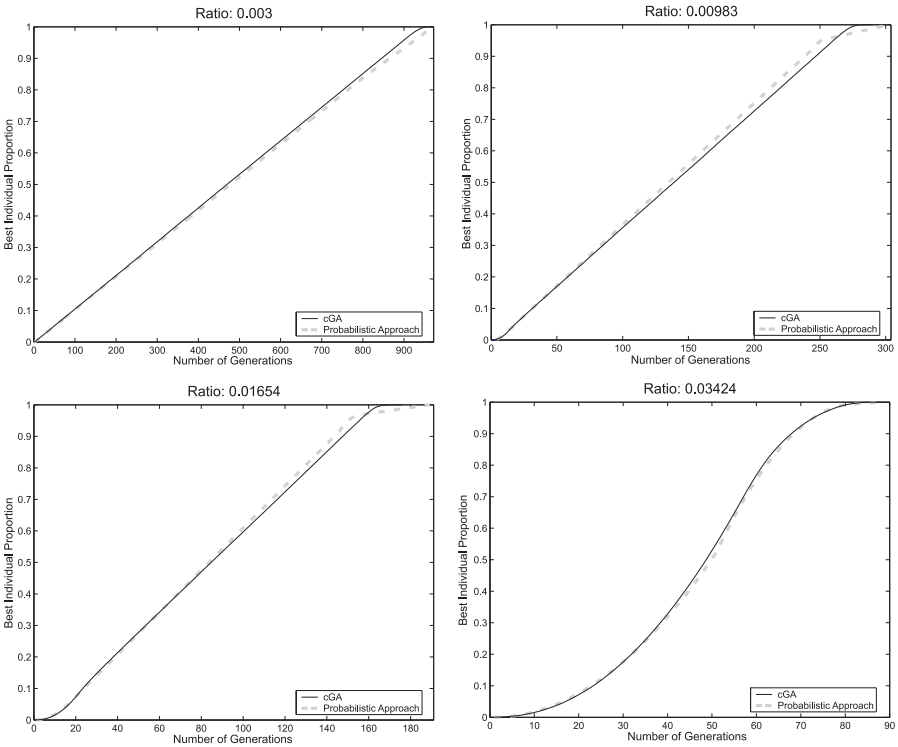


Fig. 4.8. Probabilistic approach to the convergence curves of four cGAs (binary tournament selection)

Linear Ranking Selection

In linear ranking selection, all the individuals of the neighborhood are ordered in a list according to their fitness values (from best to worst). The probability of selecting the parent is higher as the ranking in the list is better. Concretely, the probability of selecting an individual i for the replacement step is:

$$p_i = \frac{2 \cdot (n - j)}{n \cdot (n - 1)} \ , \tag{4.5}$$

where n is the size of the neighborhood (the length of the ranking) and j is the position of the individual i in the ranking.

For the linear ranking selection, we define the probabilities of selecting a high fitness individual from the neighborhood as $p_1 = 2/5$ and $p_2 = 7/10$ in case of having one or two high fitness individuals in the neighborhood, respectively.

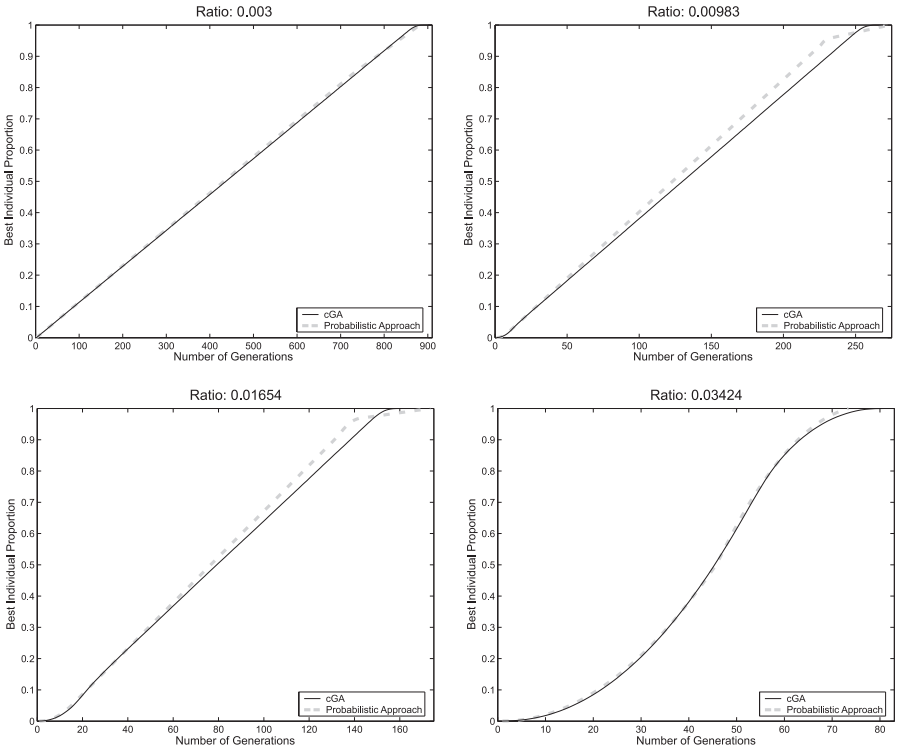


Fig. 4.9. Probabilistic approach to the convergence curves of four cGAs (linear ranking selection)

Like in the binary tournament case, the approximation is not as good as in roulette wheel. The reason is the difficulty of modelling the non deterministic behavior of the cGA. The approaching graphs of the model to the cGA for the four populations studied in the two previous sections are shown in Fig. 4.9. The mean error obtained for this case is $\Delta = 4.0705 \cdot 10^{-4}$, it is in the same order of magnitude than the obtained with binary tournament and two higher orders of magnitude than the roulette wheel case.

4.2.3 Comparison of the Main Existing Mathematical Models

In this section we compare the probabilistic model studied before versus the other fitting techniques existing in the literature. The main models proposed in the literature up to now are shown in Table 4.2. They are the logistic model proposed by Sarma and De Jong [220], the hypergraph model of Sprave [234], the mathematical model by Giacobini et al. [102], and the probabilistic model proposed by Dorronsoro in his Ph.D. thesis [72].

Table 4.2. Comparison of the main mathematical models using binary tournament selection

Model	Error (Δ)	Reference
Logistic	$2.4303 \cdot 10^{-2}$	[220]
Hypergraph	$4.2969 \cdot 10^{-2}$	[234]
Giacobini et al.	$7.6735 \cdot 10^{-4}$	[102]
Probabilistic	$2.5788 \cdot 10^{-4}$	[72]

Sarma and De Jong performed in [220] a detailed empirical analysis of the effects of the neighborhood size and shape for several local selection algorithms. They proposed a simple quantitative model for cellular EAs based on the logistic family of curves already known to work for panmictic EAs [106]. The proposed function is shown in Eq. 4.6, where a is a growth coefficient and $P(t)$ is the proportion of the best individual in the population at time step t. This model threw accurate results for synchronous updates of square shaped cellular EAs, but it does not work properly with rectangular populations, as it can be seen in Table 4.2.

$$LOG(t) = \frac{1}{1 + \left(\frac{1}{P(0)} - 1\right) \cdot e^{-at}} \ . \tag{4.6}$$

The hypergraph model proposed by Sprave [234] is a unified description for any non-panmictic population structured EA, that could even end in an accurate model for panmictic populations (since they can be considered as fully connected structured populations). The population structure is modelled by means of hypergraphs. A hypergraph is an extension of a canonical graph, and the basic idea of a hypergraph is the generalization of edges from pairs of vertices to arbitrary subsets of vertices. Sprave developed a method to estimate growth curves and takeover times of EAs using the hypergraph model. This method is based on the calculation of the diameter of the actual population structure and on the probability distribution induced by the selection operator.

The method by Giacobini et al. [102] is a probabilistic model for fitting the behavior of cGAs with square populations and different synchronous and asynchronous update policies. The authors also address with their models the prediction of takeover regimes for cEAs whose population shape is toroidal but not square, and they test them for the binary tournament selection.

Finally, the probabilistic model by Dorronsoro [72] was designed for fitting the selection pressure curves of different synchronous cGAs with any ratio value. This model was tested on the main three selection schemes that can be found in the literature.

As it can be seen in Table 4.2, the probabilistic model by Dorronsoro [72] is the most accurate one of the compared models. The error obtained by the model of Giacobini et al. [102] is in the same order of magnitude, and the logistic and the hypergraph approaches are two orders of magnitude worse than the two compared probabilistic models.

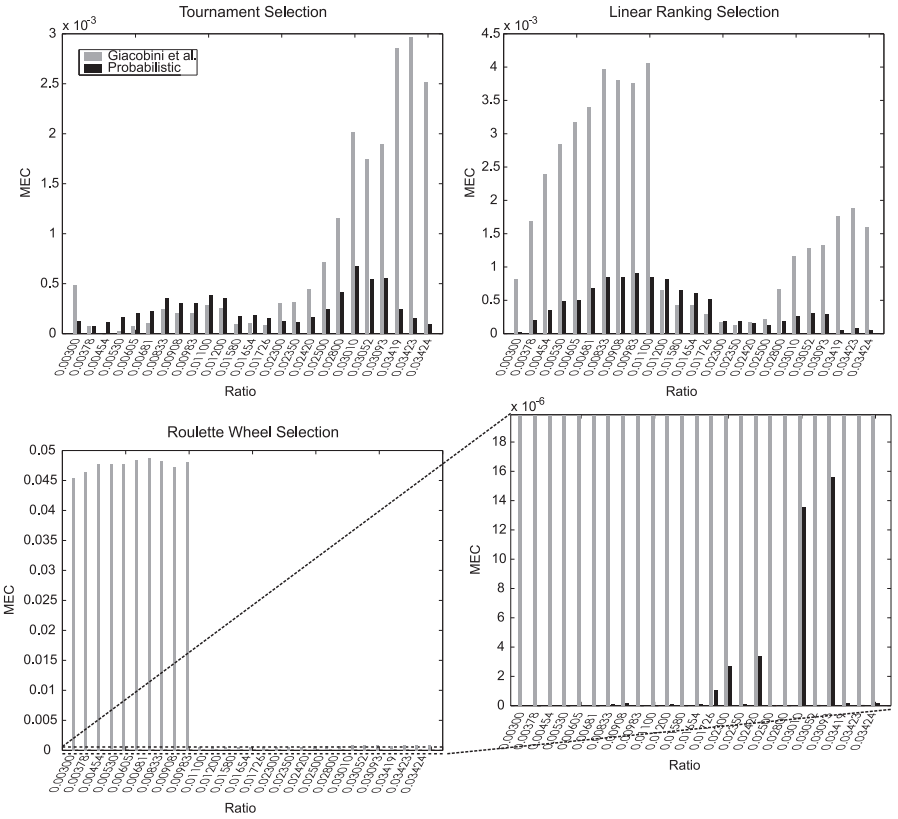


Fig. 4.10. Comparison of the obtained errors by the Giacobini et al. model and the probabilistic one presented in this book

Next, we proceed to compare the two best models analyzed. That is, the Giacobini et al. model and the probabilistic approach proposed by Dorronsoro. Therefore, in Fig. 4.10 it is shown a comparison between the two models by means of the mean square error –or MSE, defined in (Eq. 4.3)–. In that figure, 75 different cGAs (using three different selections methods and 25 population shapes) are studied. The value of ratio = 0.003 refers to the narrowest studied population in this work, composed by 4×1024 individuals, meanwhile the ratio value 0.3424 corresponds to the square population (64×64 individuals).

The three graphs plotted in Fig. 4.10 show the superiority of the probabilistic model of Dorronsoro compared to the one by Giacobini et al. It is striking the large difference existing between the two models when using roulette wheel selection. Anyway, the Giacobini et al. model is able to improve the error obtained by the probabilistic model of Dorornsoro for some intermediate ratios with binary tournament and linear ranking selections, although the differences are negligible in most cases. The mean error (Δ) incurred by each

Table 4.3. Comparison between the model of Giacobini et al. and the probabilistic one proposed in this work in terms of the mean error (Δ)

Selection	Giacobini et al.	Probabilistic
Roulette	0.0194	$\mathbf{1.4904 \cdot 10^{-6}}$
Tournament	$7.6735 \cdot 10^{-4}$	$\mathbf{2.5788 \cdot 10^{-4}}$
Linear Ranking	0.0017	$\mathbf{4.0705 \cdot 10^{-4}}$
Mean Value	$7.2891 \cdot 10^{-3}$	$\mathbf{2.2214 \cdot 10^{-4}}$

model for the three studied selections and the mean of the three cases are shown in Table 4.3. As it can be seen, the model by Dorronsoro is the best one for the three selections (see **bolded** figures in Table 4.3), being the mean error of the three selections nearly two orders of magnitude lower than in the model of Giacobini et al.

If we observe the behavior of the model proposed by Giacobini et al., it is surprising its irregular fitting dependent of the selection method used. For instance, this model behaves clearly bad for large ratios in the binary tournament case, while in the case of using linear ranking this difference is much smaller (the fitting is better) for the largest ratios, but its performance importantly worsens for the smallest ones (narrowest populations).

4.3 Validation of the Theoretical Models

As it was shown in the first part of this chapter, there exists a number of works which propose some mathematical equations for modelling the selection pressure of cGAs in order to characterize their theoretical behavior.

In this section we make an empirical study for sustaining the obtained results in the previous theoretical work, where it was shown that the use of different asynchronous update policies and distinct ratios influences the behavior of the cGA. Thus, we present the results of the comparison of synchronous and asynchronous cGAs, and also the results of synchronous cGAs with different ratio values, using always a constant neighborhood shape (NEWS). Notice that the complete study of the problems of this section is out of the scope of this work, and it is not the goal of this study to compare the performance of the studied cGAs versus the state-of-the-art algorithms and other heuristics for combinatorial and numerical optimization. If this were the case, we should at least, tune the parameters and include a local search step in the algorithm. Therefore, the results presented in this section belong only to the relative performance of the cGAs with different update policies and ratios among them.

Section 4.3.1 is devoted to the discrete case, while in Sect. 4.3.2 we focus on continuous optimization.

Table 4.4. Parameterization used in the studied cGAs for discrete problems

Population size	400 individuals
Neighborhood	NEWS
Parent selection	Binary tournament + binary tournament
Recombination	DPX, $p_c = 1.0$
Bit mutation	Bit-flip, $p_m = 1/L$ ($10/L$ for FMS) (L = Individual length)
Replacement	Rep_if_Better

4.3.1 Validation on Combinatorial Optimization

In this section we present and analyze the results of solving a set of discrete problems with asynchronous cGAs using diverse individual update policies, and other synchronous cGAs with different population grid shapes.

In our experiments, we work with the same set of problems studied as in [26], which includes the massively multimodal deceptive problem (MMDP), the frequency modulation sounds problem (FMS), and the multimodal problems generator P-PEAKS. Moreover, we have extended this set of problems with the error correcting codes design problem (ECC), the maximum cut of a graph (MAXCUT), the minimum tardy task problem (MTTP) and the satisfiability problem (SAT). All of them are described in Appendix A. The size of the MDDP problem used in this study is $k = 20$, meanwhile MTTP20 and MAXCUT20.09 are the studied instances of MTTP and MAXCUT, respectively.

We can consider the selected set of problems for this study representative, as it contains many interesting characteristics, like multimodality, the use of constraints or the identification of parameters; moreover, most of them are deceptive functions or problem generators. These are important elements in any work evaluating different algorithmic approaches with the goal of obtaining reliable results, as it is declared by Whitley et al. in [261].

The election of this benchmark is justified by both the high difficulty and the domains of application (parameter identification, telecommunications, combinatorial optimization, scheduling, etc.). This provides results with a high level of relevance, although the evaluation of the conclusions is consequently more difficult than when a reduced set of problems is used. Next, we present and analyze the obtained results during our tests.

Experimental Analysis

Next, we present the obtained results after solving the previously proposed problems with four asynchronous cGAs and three synchronous cGAs with different static ratios. The configuration of the algorithm for the combinatorial optimization case is shown in Table 4.4. As we can see, the population is always composed of 400 individuals. The neighborhood used is NEWS, where the two

Table 4.5. Studied ratios

Name	Population Shape	Ratio
Square	20×20 individuals	0.110
Rectangular	10×40 individuals	0.075
Narrow	4×100 individuals	0.031

parents are selected using binary tournament. The recombination operator used is the two points crossover and the mutation lies in changing the value of each bit with a probability of $1/L$, being L the length of the individuals. Finally, the new generated individual (offspring) replaces the current one only if it is better, that is, if the fitness value is higher (assuming we want to maximize). The termination condition of the algorithm is either to find the optimal solution or to execute a maximum number of generations of the algorithm. This maximal number of generations is different for each problem (due to their different complexities) and it is specified in captions of Tables 4.6 to 4.12.

In Table 4.5 we present the ratios used in this study. As we can see, they are a square population, with ratio = 0.11 (Square), and two rectangular populations: of 10×40 (Rectangular) and 4×100 individuals (Narrow), with ratio values of 0.075 and 0.031, respectively.

In the next tables we show the obtained results for the previously mentioned problems: MMDP (Table 4.6), FMS (Table 4.7), P-PEAKS (Table 4.8), ECC (Table 4.9), MAXCUT20.09 (Table 4.10), MTTP20 (Table 4.11), and SAT (Table 4.12). In these tables we present the average values of the best fitness found in each execution, the necessary average number of evaluations for obtaining the optimal solution to the problem (if it is found), and the percentage of executions in which the optimum is found (hit rate). Therefore, we are analyzing the final distance to the optimum (specially interesting when the optimum is not found), the effort needed by the algorithm, and the expected efficacy of the algorithm, respectively. In order to obtain significant results, we performed 100 independent runs of each algorithm for all the studied problems. The best results for each problem are **bolded**.

Analyzing these tables we can obtain some clear conclusions. First, the asynchronous studied algorithms tend to need a lower number of evaluations than the synchronous ones for obtaining the optimal solution, in general. Moreover, differences between synchronous and asynchronous algorithms are statistically significant for all the cases (except in two of them: MMDP and SAT), which indicates that the asynchronous versions are more efficient than the synchronous cGAs with different ratios. This result confirms the theoretical one in which it was shown that the asynchronous cGAs perform higher selection pressures than the synchronous ones.

On the other hand, the synchronous algorithms obtain a similar percentage of optimal solutions found (hit rate) or even better than the asynchronous versions, while there not always exist significant differences in the quality of the solutions found (the exceptions are, probably, due to the differences in the hit rate).

Table 4.6. MMDP problem with a maximum of 1000 generations

	Algorithm	Mean Solution (Optimum=20)	Mean Number of Generations	Hit Rate
Synch.	Square	19.813	214.2	57%
	Rectangular	19.824	236.1	58%
	Narrow	19.842	299.7	**61%**
Asynch.	LS	19.518	343.5	23%
	FRS	19.601	209.9	31%
	NRS	19.536	**152.9**	28%
	UC	**19.615**	295.7	36%
	Test	+	+	

Table 4.7. FMS problem with a maximum of 3000 generations

	Algorithm	Mean Solution (Optimum=100)	Mean Number of Generations	Hit Rate
Synch.	Square	**90.46**	437.4	57%
	Rectangular	85.78	404.3	61%
	Narrow	80.76	610.9	**63%**
Asynch.	LS	81.44	**353.4**	58%
	FRS	73.11	386.2	55%
	NRS	76.21	401.5	56%
	UC	83.56	405.2	57%
	Test	+	+	

Table 4.8. P-PEAKS problem with a maximum of 100 generations

	Algorithm	Mean Solution (Optimum=1)	Mean Number of Generations	Hit Rate
Synch.	Square	**1.0**	51.8	**100%**
	Rectangular	**1.0**	50.4	**100%**
	Narrow	**1.0**	**53.9**	**100%**
Asynch.	LS	**1.0**	34.8	**100%**
	FRS	**1.0**	38.4	**100%**
	NRS	**1.0**	38.8	**100%**
	UC	**1.0**	40.1	**100%**
	Test	−	+	

If we now pay attention to the obtained hit rates, we can conclude that the synchronous policies improve the asynchronous ones: slightly in the case of the fitness values found and clearly in terms of the probability of finding the optimal solution (that is, the frequency of finding an optimum).

Another interesting result is the fact that we can define two kinds of problems: those in which the optimal solution is found by all the cGAs (100% hit rate), and those others for which the optimum can not be found in the 100% of the runs by any algorithm. Problems that seem to be affordable directly with cGAs, or problems for which the cGAs need some help (not studied yet), as including local search for example.

Table 4.9. ECC problem with a maximum of 500 generations

	Algorithm	Mean Solution (Optimum=0.0674)	Mean Number of Generations	Hit Rate
Synch.	Square	0.0670	93.9	85%
	Rectangular	0.0671	93.4	88%
	Narrow	**0.0673**	104.2	**94%**
Asynch.	LS	0.0672	79.7	89%
	FRS	0.0672	82.4	90%
	NRS	0.0672	**79.5**	89%
	UC	0.0671	87.3	86%
	Test	+	+	

Table 4.10. MAXCUT20.09 problem with a maximum of 100 generations

	Algorithm	Mean Solution (Optimum=56.74)	Mean Number of Generations	Hit Rate
Synch.	Square	**56.74**	11.3	**100%**
	Rectangular	**56.74**	11.0	**100%**
	Narrow	**56.74**	11.9	**100%**
Asynch.	LS	**56.74**	**9.5**	**100%**
	FRS	**56.74**	9.7	**100%**
	NRS	**56.74**	9.6	**100%**
	UC	**56.74**	9.6	**100%**
	Test	−	+	

Table 4.11. MTTP20 problem with a maximum of 50 generations

	Algorithm	Mean Solution (Optimum=0.02439)	Mean Number of Generations	Hit Rate
Synch.	Square	**0.02439**	8.4	**100%**
	Rectangular	**0.02439**	8.3	**100%**
	Narrow	**0.02439**	8.9	**100%**
Asynch.	LS	**0.02439**	**5.9**	**100%**
	FRS	**0.02439**	6.2	**100%**
	NRS	**0.02439**	6.3	**100%**
	UC	**0.02439**	6.3	**100%**
	Test	−	+	

In order to summarize the large set of results generated and obtain practical conclusions, we present a ranking of the best algorithms according to three different metrics: mean of the best solution found, mean of the number of generations needed for finding the optimum (when it is found), and the hit rate. These three rankings, which are shown in Table 4.13, have been computed by adding the position (from best to worst: 1, 2, 3, ...) where the algorithms are placed for the previous results presented from Table 4.6 to Table 4.12, according to the three criteria. The value of this sum of positions is also shown in Table 4.13 for every algorithm and criterion.

Table 4.12. SAT problem with a maximum of 3000 generations

	Algorithm	Mean Solution (Optimum=430)	Mean Number of Generations	Hit Rate
Synch.	Square	429.54	703.1	79%
	Rectangular	**429.67**	706.3	84%
	Narrow	429.61	763.7	81%
Asynch.	LS	429.52	**463.2**	78%
	FRS	**429.67**	497.7	**85%**
	NRS	429.49	610.5	75%
	UC	429.50	725.5	76%
	Test	−	+	

Table 4.13. Ranking of the best algorithms for discrete problems

Average Solutions			Average Generations			Hit Rate		
Algorithms Ranking		Sum	Algorithms Ranking		Sum	Algorithms Ranking		Sum
1	Narrow	10	**1**	LS	14	**1**	Narrow	6
1	Rectangular	10	**2**	NRS	16	**2**	Rectangular	10
3	Square	14	**3**	FRS	18	**3**	FRS	14
4	FRS	15	**4**	UC	30	**4**	LS	15
5	LS	18	**5**	Rectangular	33	**5**	Square	17
5	UC	18	**6**	Square	37	**6**	UC	19
7	NRS	21	**7**	Narrow	48	**7**	NRS	21

As we expected after the previous results, according to the mean of the best solution found and the hit rate criteria, the synchronous algorithms with any of the three studied ratios are in general more accurate than all the asynchronous ones for the problems used in our tests, with a specially privileged position for the Narrow population. The reason is that these synchronous algorithms induce a lower selection pressure than the asynchronous ones, as we previously proposed in Sect. 4.1, so they have a more explorative behavior. This allows them (with respect to an equivalent asynchronous cGA) both to explore a wider region of the search space and to scape from local optimal solutions. On the other hand, the asynchronous versions clearly outperform the synchronous algorithms in terms of efficiency, i.e., the mean of generations needed for finding the solution (they have a higher selection pressure), tending towards LS as the best asynchronous policy for discrete problems.

4.3.2 Validation on Continuous Optimization

In this section we extend the work made in the previous one through the application of the same algorithmic models to some continuous functions in order to obtain a more extensive study. This study may be interesting for analyzing the behavior of the algorithms in continuous optimization, with the possibility of comparing the results with the discrete case. The functions we have

Table 4.14. Parameterization used in the studied cGAs for continuous problems

Population size	400 individuals
Neighborhood	NEWS
Parent selection	Binary tournament + binary tournament
Recombination	AX, $p_c = 1.0$
Mutation of genes	Uniform, $p_m = 1/2L$ (L = Individual length)
Replacement	Rep_if_Better

selected for this study are three typical multimodal functions in benchmarks of numerical problems: they are the Rastringin (with dimension 10), Ackley and Fractal functions (see Appendix A for a description of the problems). For solving these three problems, we have used real coded individuals, meanwhile in the previous selection we used binary coded individuals. This is the reason of our special interest in the experimentation with a more traditional global optimization. The codification we used for these three problems follows the implementation proposed by Michalewicz [181].

Experimental Analysis

In this section we study the results of applying our algorithms to the proposed continuous problems, as we did before in the discrete case. We maintain the same ratios used in the synchronous algorithms studied in Sect. 4.3.1, as well as the asynchronous policies. On the other hand, we need a especial configuration for genetic operators and for their application probabilities in these real-coded problems. This parametrization is detailed in Table 4.14. As we can see, the population size, the neighborhood, the selection policy of the parents and the replacement used in this case are the same than in the discrete case. However, the recombination and mutation operators used change, they are operators typically used in continuous optimization, namely the arithmetic crossover (AX) and the uniform mutation [181].

We present from Table 4.15 to 4.17 the results of our experiments with the Rastrigin (Table 4.15), Ackley (Table 4.16), and Fractal (Table 4.17) problems. As in the case of discrete problems, these tables contain the mean of the best solutions found, the mean number of generations needed to find the optimum, and the hit rate. These three values have been calculated on 100 independent runs. For these three real codification problems, we consider that the search successfully finishes when an individual reach the optimum with a 10% error.

The results obtained with continuous problems are not as clear as in the discrete case. According to the mean number of generations needed to find an optimal solution, the asynchronous algorithms (with higher selection pressure) do not always achieve a lower number of generations than the synchronous ones; two examples are the Ackley and Fractal functions. The differences between synchronous and asynchronous algorithms are significative in general. It indicates a higher efficiency of the cGAs with different ratios, what contrasts to the conclusions obtained from the theory. However, we think that the dif-

Table 4.15. Rastrigin problem with a maximum of 700 generations

	Algorithm	Mean Solution (Optimum$\leq$ 0.1)	Mean Number of Generations	Hit Rate
Synch.	Square	0.0900	323.8	**100%**
	Rectangular	0.0883	309.8	**100%**
	Narrow	**0.0855**	354.2	**100%**
Asynch.	LS	0.0899	**280.9**	**100%**
	FRS	0.9000	289.6	**100%**
	NRS	0.0906	292.2	**100%**
	UC	0.0892	292.4	**100%**
	Test	$-$	$+$	

Table 4.16. Ackley problem with a maximum of 500 generations

	Algorithm	Mean Solution (Optimum$\leq$ 0.1)	Mean Number of Generations	Hit Rate
Synch.	Square	0.0999	321.7	78%
	Rectangular	0.0994	293.1	73%
	Narrow	0.1037	**271.9**	65%
Asynch.	LS	**0.0932**	302.0	84%
	FRS	0.0935	350.6	**92%**
	NRS	0.0956	335.5	87%
	UC	0.0968	335.0	85%
	Test	$+$	$+$	

Table 4.17. Fractal problem with a maximum of 100 generations

	Algorithm	Mean Solution (Optimum$\leq$ 0.1)	Mean Number of Generations	Hit Rate
Synch.	Square	0.0224	75.2	94%
	Rectangular	0.0359	62.8	78%
	Narrow	0.1648	**14.6**	16%
Asynch.	LS	0.0168	69.7	98%
	FRS	0.0151	71.5	**100**%
	NRS	0.0163	73.6	98%
	UC	**0.0138**	72.8	96%
	Test	$+$	$+$	

ferences on the hit rates reported by the algorithms explain this result. In fact, in contrast to what we observed in the discrete case, the hit rates obtained by the asynchronous algorithms are higher than those of the synchronous cGAs, in general. This is probably because the maximum allowed number of generations is too low for the synchronous cGAs, which cannot converge as fast as the synchronous ones (the selection pressure of the asynchronous algorithms is lower). Finally, opposite to the results in Sect. 4.3.1, where either all the algorithms find the optimum in the 100% runs or no algorithm can find it, the asynchronous cGA using the FRS update policy is the single algorithm able to find the optimal solution in all the executions of the Fractal problem.

Table 4.18. Ranking of the best algorithms for continuous problems

Average Solutions			Average Generations			Hit Rate		
Algorithms Ranking		Sum	Algorithms Ranking		Sum	Algorithms Ranking		Sum
1	UC	8	**1**	LS	7	**1**	FRS	3
2	LS	9	**2**	Narrow	9	**2**	NRS	5
2	FRS	9	**2**	Rectangular	9	**3**	LS	7
4	NRS	13	**4**	FRS	13	**4**	UC	8
4	Rectangular	13	**5**	UC	14	**5**	Square	11
6	Narrow	15	**6**	NRS	15	**6**	Rectangular	13
7	Square	16	**7**	Square	17	**7**	Narrow	15

In order to summarize the results, and following the structure of Sect. 4.3.1, we present in Table 4.18 a ranking of the best algorithms for all the problems in terms of the mean of the best solution known, the mean number of generations needed to find the optimum and the hit rate. In this table we can see the trend of the asynchronous algorithms to be better than the synchronous ones in terms of the mean of the best solution found and the hit rate. On the other hand, the synchronous algorithms with different ratios seem to be more efficient than most of the asynchronous ones in general (the exceptions are the synchronous cGA with square grid –Square– and the asynchronous LS).

4.4 Conclusions

In the first part of this chapter we studied the selection pressure induced by some asynchronous update policies of individuals in cGAs, and also by different synchronous cGAs with distinct ratio values (different shapes of the population). All these studied algorithms perform a different search in the solution space. As we can see, it is possible to fix the selection pressure of a cGA by choosing an update policy and/or a ratio value between the grid of population and the neighborhood without the necessity of tuning the additional numerical parameters of the algorithm. This is a main issue of the algorithms, as the user can use the existing knowledge instead of creating a new heuristic class.

Our conclusion is that cGAs can be easily induced to promote the exploration or exploitation by simply changing the update policy or the ratio between the population and the neighborhood. This opens new research lines to decide new efficient ways of changing from a given policy or ratio to others that reach the optimal solution with lower effort than in the case of the canonical cGA or other kinds of GAs.

In the second part of this chapter we present some mathematical models which allow us to quantify, in advance, the selection pressure of cEAs through mathematical equations. We have studied the main existing models in the literature, and compared the two best ones. The best performing methods are those models that are decomposed into three different equations.

Finally, in the third part of this chapter we applied our extended cGAs to a set of problems belonging to the combinatorial and continuous optimization domains. Although our aim was not to obtain techniques able to compete with specialized heuristics of the state of the art, the results clearly show that cEAs are very efficient optimization techniques that can be improved even more by hybridizing them with local search techniques [14, 15, 16, 92]. The results of the test problems clearly confirm, with some little exceptions, that the capacity of these algorithms for solving the problems by using different ratios and update policies is directly related to the selection pressure of the algorithms, showing that exploitation plays a very important role in the search. It is clear that the role of exploration should be more important in more difficult problems than the studied ones, but we can address this issue in our algorithms using a more explorative adjustment of the parameters, and using different strategies of cEAs in different times during the search in a dynamical way [10, 12].

Our results are very clear: regarding the discrete problems, the asynchronous algorithms are more efficient than the synchronous ones; with statistically significant differences in most problems. On the other hand, if we pay attention to the other two studied issues, we can conclude that the synchronous cGAs with different ratios outperform the asynchronous algorithms: slightly in terms of the mean of solutions found, and clearly in terms of the probability of finding the solution (that is, the frequency of finding the optimum).

On the contrary, in the case of the experiments in continuous problems, we can obtain different conclusions (complementary in some way). The asynchronous algorithms improve the synchronous ones according to the mean solutions found and hit rate, while the synchronous algorithms are in general more efficient than the asynchronous ones in terms of the mean number of generations made.

Part III

Algorithmic Models and Extensions

5

Algorithmic and Experimental Design

God does not care about our mathematical difficulties. He integrates empirically.

Albert Einstein (1879 - 1955) – Physicist

The main hypothesis in this book is that we can create better algorithms by relying to cellular concepts in GAs. To this end, we have followed the usual scientific way of *analyzing, proving, and modelling* concepts. Now we arrive to the point of depicting how the rest of hypothesis support is going to be developed: simulation and experimentation. In the big picture, a little bit of all this has been used in the previous chapters, but it is in this chapter where the whole abstract approach is sketched.

After the introduction to the cEAs field and the study of the state of the art in this domain performed in previous chapters (Chaps. 4 and 2, respectively), we present here some of the most important improvements to the proposed canonical cEA existing in the literature. These new models aim to improve the efficiency, accuracy and/or efficacy of the canonical cEA by modifying the exploration/exploitation tradeoff performed by the algorithm on the search space in some way.

We describe in Sect. 5.1 the new approaches for improving the behavior of the canonical cGA that are addressed in this book. In Sect. 5.2, it is shown the way in which we proceed in our studies for obtaining the reported results, and how they are analyzed in order to obtain statistical relevance in our conclusions when comparing the different models. Finally, we present in Sect. 5.3 our main conclusions.

5.1 Proposal of New Efficient Models

Nowadays, there exists a huge interest for finding more efficient optimization techniques without losing their capacity for being applied to new problems. As it was already said, we focus on cellular genetic algorithms (cGAs) in this book, a kind of very efficient GA with structured population which has been explored by the scientific community in lesser extent than other GAs as panmictic or distributed ones. Despite of that, cGAs are techniques which have demonstrated to be very efficient when solving really complex problems

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_5, © Springer Science+Business Media, LLC 2008

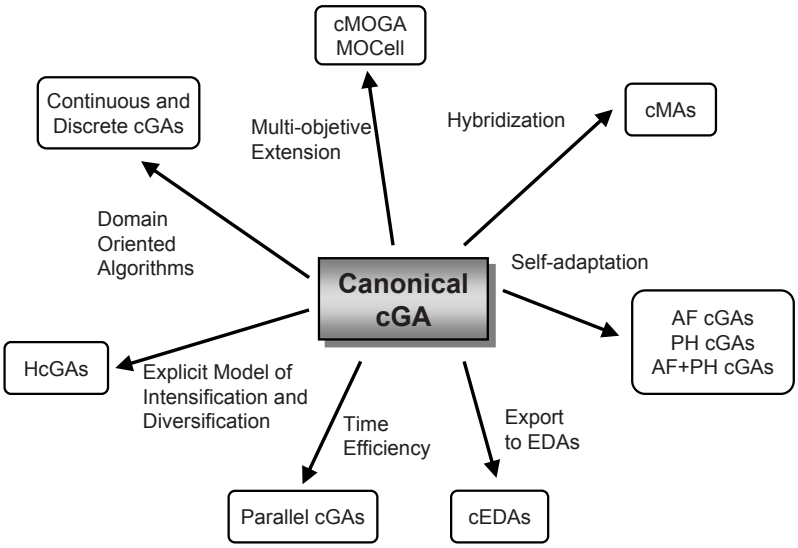


Fig. 5.1. New algorithmic models of cGAs and extensions to other domains presented and studied in this book

and, as it was probed in Chap. 3, the use of cGAs leads us many times to obtain a better efficiency according to the equivalent structured in islands and panmictic GAs.

This book mainly focuses on the search of new models of cGAs which improve the behavior of an equivalent canonical cGA, and on the adaptation of the cellular model to other domains where they have not been exported until the moment, in order to improve the existing proposals in these domains. There are too many ways for achieving this goal. For example, in the recent literature of cGAs we can find some examples which try to improve the efficacy of these algorithms investigating new genetic operators [229], using different population topologies [101, 203], adjusting the tradeoff between the exploration and the exploitation performed by the algorithm in the search space [26, 192], etc.

In this book, we pay special attention to those algorithms of innovative design (see Fig. 5.1) which improve the tradeoff between exploration and exploitation performed on the search space (e.g., hierarchial cGAs, adaptive cGAs, or cellular memetic algorithms), on the adaptation of some of the developed models to new domains where cGAs have not been applied, and on the search of efficient parallel models.

We think that a really appropriate methodology for improving the behavior of an heuristic is working on its principles of functioning. For that, it is necessary to study and characterize theoretically the algorithm, although many researchers do not consider this important step. However, we performed this study in our investigations (see Chap. 4). Maybe, the most influential feature

on the behavior of an optimization algorithm is the tradeoff between exploration and exploitation applied on the search space. These are two opposite characteristics (that is, they are in conflict each other) which coexist in the algorithm for achieving the success. As we said before, an heuristic with a high exploration capacity will soon converge to a local optimum, from where surely it will not be able to scape. On the contrary, if the algorithm promotes the exploration of the search space too much, it will be able to explore an important area of the solution space of the problem, but it can not deepen in any region, so probably, it will find low quality solutions. Thus, an appropriate balance between exploration and exploitation allows the algorithm to wander a wide area of the search space (exploration), and to deepen in the most promisingly areas (exploitation). So, it is desirable that the algorithm keeps the exploration capacity in every moment as it is the responsable of allowing the heuristic to scape from local optima.

For this reason we made an effort in the theoretical study on the behavior of the cGAs, and why we present some new models which try to improve the balance between exploration and exploitation of the algorithms. In this way, we intend to develop new algorithmic improvements for the canonical model of a cGA which allow us to outperform its behavior. The proposed extensions follow different general philosophies for achieving efficient and accurate algorithms at the same time. Moreover, these algorithmic improvements are frequently accumulative, meaning for example that we can combine the advantages of two of our proposals in other new algorithms in the future (for instance, hierarchical cEDAs, adaptive cEDAs or multi-objective parallel cGAs). The algorithmic models proposed are summarized below:

- **Adaptive cGAs.** In Chap. 6 we propose three new algorithms which automatically regulate the tradeoff between the exploration and exploitation according to the convergence speed of the population. In this way, if the population converges too fast, the exploration of the algorithm is promoted for slowing down the lost of diversity, whereas if the convergence is slow, the local exploitation of the solutions will be emphasized. Three different methods has been used for measuring the convergence speed of the population, obtaining the three cGAs proposed: AF cGA, where the convergence speed is measured according to the average fitness value of the individuals in the population; PH cGA, where the entropy of the population is used; and AF+PH cGA, which uses both values, average fitness and entropy.
- **Hierarchical cGAs.** In Chap. 10 a new model of cGA is presented for improving the tradeoff between exploration and exploitation of the canonical cGA. In this case, a hierarchy is established in the population by placing the solutions in a piramidal disposition, where the best individuals are located in the center of the population (lowest level), and the farther from the center (rising levels) the worse the individuals are. With this hierarchy we promote the exploitation of the best solutions, meanwhile the diversity of the solutions is maintained in the higher levels.

- **Cellular EDAs.** We think that it can be interesting to export the advantages of the cellular model to other families of EAs different from the GAs. In Chap. 10, we address a new algorithmic model of EDA, where the population is structured in a toroidal grid. The result is the first cellular EDA in the literature.
- **Cellular memetic algorithms (cMAS).** Memetic algorithms are highly specialized techniques. They use information from the problem to solve in different parts of the algorithm, as the individuals representation or the variation operators, for instance. Moreover, they usually include a local search method for helping to refine the solutions. In Chap. 7 we present the first cMA in the literature, up to our knowledge. The developed algorithm was applied to the SAT problem (a description of the problem is available in Appendix A). Additionally, we propose the study of other cMAs for the vehicle routing and the DNA fragment assembly problems in Chaps. 13 and 15, respectively.
- **Parallel cGAs.** In Chap. 8 we propose two new parallel models of cGAs. One of them is designed for running in local area networks and it is loyal to the canonical cGA model, whereas the other one is a model distributed in islands, in which cGAs are executed. This second model is designed for *grid computing* environments.
- **Multi-objective cGAs.** We present two proposals of cGAs for the multi-objective optimization domain: MOCell (Chap. 9) and cMOGA (Chap. 14). MOCell is an evolution from cMOGA which achieve better results, but it is not ready (yet) for working with heterogeneous individuals (with genes of different types), as it is needed in the problem tackled with cMOGA in Chap. 14.
- **cGAs in continuous domains.** Finally, thanks to the characteristics of GAs, we can apply our algorithm to some other optimization domains simply defining a new kind of individual and operators for this domain. In Chap. 12 we exploit this property applying JCell to the continuous optimization domain.

5.2 Evaluation of the Results

As it is known, GAs belong to the field of metaheuristics and thus they are non-deterministic techniques. This implies that two executions of the same algorithm on a given problem can reach different solutions. This particular characteristic of metaheuristics supposes an important problem for the researchers when evaluating the results and, therefore, when comparing their algorithms to other existing ones.

There exist some works which tackle the theoretical analysis for a large number of heuristics and problems [122, 142] but, due to the difficulty of this theoretical analysis, the behavior of the algorithms are analyzed traditionally through empirical comparisons. For that, the studied techniques are applied to

a wide rank of well known problems of diverse characteristics. In this way, the algorithms can be compared according to the quality of the solutions obtained and the computational effort used to find them. This effort is calculated in terms of the number of evaluations of the fitness function performed during the search.

Along the studies made in this book, we always tried to follow the same methodology, which can be divided in three stages or main steps:

1. The first point consists of the definition of the goals we want to achieve, the problem instances, and the issues in which we focus for evaluating and analyzing our results.
2. The second step is to define the metrics selected for measuring the results, to execute the experiments for obtaining these results, and to analyze the obtained data. In Sects. 5.2.1 and 5.2.2 the metrics used in our work, and the methodology used for analyzing the results are presented.
3. Finally, we consider it is very important to inform about the mechanism followed during the experiments, in order to be easily reproducible by other researchers. So that, it is necessary to provide all the information related to the used algorithms (code, parameters, etc.), to the studied problems (instances, size, characteristics, references, etc.), and even to the execution environment (memory, processor, operating system, programming language, etc.). It is really important how all the information of a work is presented. This information must be easily assimilable by the reader without penalizing the clarity of the contents, for avoiding any confusion.

Some of the studies developed in this work involve the use of multi-objective problems, which must optimize more than one function simultaneously (the algorithms used for these kind of problems are called multi-objective algorithms). Moreover, the functions composing a multi-objective problem are usually in conflict, what means that improving one of the objectives implies worsen any of the rest. Therefore, the solutions to this kind of problems are not a single value, but a set of non-dominated points (consult Chap. 9 for more details). It is necessary, hence, the use of metrics for comparing different results of this kind of problems. The application of these metrics allows us to compare the algorithms, and even to apply the statistic analysis previously commented for obtaining statistic confidence in our conclusions. In Sect. 5.2.2 the metrics used in this book for the multi-objective case are presented.

5.2.1 The Mono-objective Case

Due to the non-deterministic nature of the studied algorithms, comparisons on the results of a single execution are not consistent, so they must be done on a large set of results obtained after a high number of independent executions of the algorithm performed on the given problem. For making comparisons

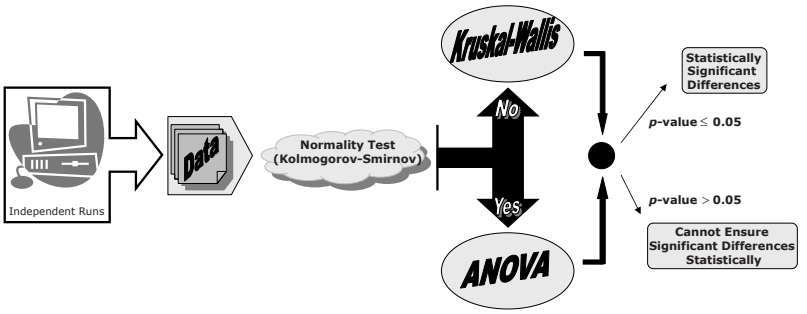


Fig. 5.2. Scheme of the evaluation process of the results

between the obtained results we must use statistic functions. In the literature, the functions traditionally used are very simple, as the average of the values obtained, the median, or the best and worst solutions found by the studied technique. However, these are not conclusive values, and they can lead us to wrong readings of the results. Hence, we must use statistic studies which guarantee the significance of results and comparisons presented in our studies.

In order to obtain results with *statistical significance*, researchers usually use t-tests or analysis of variance (ANOVA) functions. The use of these statistic functions allows us to determine whether the effects observed in the obtained results are significative or, on the contrary, they are due to errors in the samples performed. But the use of ANOVA function on any data distribution is not really suitable, as in a non normal data distribution the result may be wrong. In this case, it would be better to apply a Kruskal-Wallis test.

In Fig. 5.2 we present the methodology applied in the evaluation and interpretation of the results obtained in the performed experiments along the works carried out for this book. For obtaining the results of our tests we performed at least 30 independent executions of the algorithm on the given problem, although in most of the studies presented here the number of independent runs increase up to 100. Once the results of the executions are obtained, a Kolmogorov-Smirnov test is applied in order to check whether the values of the results follow a normal distribution (gaussian) or not. If the distribution is normal, an ANOVA test is performed, meanwhile if not, the test applied is the Kruskal-Wallis one. In this work we always consider a confidence level of 95% (significance level of 5% or p-value under 0.05) in our statistic tests. This means we can guarantee that the differences of the compared algorithms are significative or not with a probability of 95%.

5.2.2 The Multi-objective Case

As it was previously introduced, when dealing with multi-objective algorithms we need to use metrics for comparing the quality of the solutions of different solution techniques. Once these metrics are applied, we can do the same statistical tests previously proposed for the mono-objective case to the obtained results. Currently, it does not exist any metric in the literature which

guarantees the superiority of a multi-objective algorithm versus another one for a given set of problems. Therefore, we should use several metrics which focus on diverse issues of the solutions for comparing the algorithms studied. For obtaining reliable results it is convenient to normalize the obtained results before applying these metrics. Following the recommendations proposed in [65], we apply the normalization according to the optimal front when it is known, or according to the best and worst solutions in our front if the optimal one is not known. In the works performed for this book, the metrics used are:

- **Generational Distance – GD.** This metric was introduced by Van Veldhuizen and Lamont [254] for measuring how far the elements of the set of non-dominated vectors found are from the optimal Pareto set. It is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \ , \tag{5.1}$$

  where n is the number of vectors in the set of non-dominated solutions, and d_i is the Euclidean distance (measured in the objective space) between each of the solutions and its nearest member in the optimal Pareto set. According to this definition, it is clear that a value $GD = 0$ means that all the generated elements are in the optimal Pareto set.
- **Dispersion – Δ.** The *dispersion* metric [67] is a measure of the diversity which quantifies the value the dispersion reaches between the solutions obtained. This metric is defined as:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \bar{d} \right|}{d_f + d_l + (N-1)\bar{d}} \ , \tag{5.2}$$

  where d_i is de Euclidean distance between two consecutive solutions, $\bar{d}$ is the average of these distances, and d_f y d_l are the Euclidean distances to the *extreme* solutions (limits) of the exact Pareto front in the objective space (for more details, see [67]). This measure is zero for an ideal distribution, remarking a perfect dispersion of the solutions along the Pareto front.
- **Hypervolume – HV.** The *hypervolume* metric [272], which is a combined metric of convergence and diversity, calculate the covered volume (in the objective space) by the members of a set Q of non-dominated solutions (the enclosed area by the discontinuos line in Fig. 5.3, $Q = \{A, B, C\}$) for problems where all the objectives must be minimized. Mathematically, for each solution $i \in Q$, a hypercube v_i is built using a reference point W (which can be composed by the worst solution for each aim, for example) and the solution i as the corners of the diagonal of the hypercube. The hypervolume is calculated as the volume of the union of all the hypercubes:

$$HV = volume \left(\bigcup_{i=1}^{|Q|} v_i \right) \ . \tag{5.3}$$

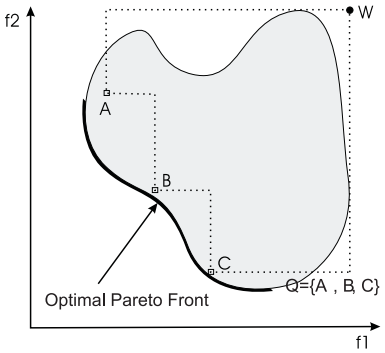


Fig. 5.3. The hypervolume of a front of non-dominated solutions

- **Number of Pareto optima.** In the resolution of some really complex multi-objective problems, finding a high number of non-dominated solutions may be a really hard task for the algorithm. In this sense, the number of Pareto optima found can be used as a measure of the capacity of the algorithm for exploring the search space defined by the multi-objective problem.
- **Set coverage – C(A, B).** The *set coverage* metric $C(A, B)$, calculates the proportion of solutions in B which are dominated by the solutions of A:

$$C(A, B) = \frac{|\{b \in B | \exists a \in A : a \preceq b\}|}{|B|} \quad . \tag{5.4}$$

  A value of the metric $C(A, B) = 1$ means that all the members of B are dominated by A, whereas $C(A, B) = 0$ means that no member of B is dominated by A. In this way, the higher $C(A, B)$, the better the Pareto front A is according to B. As de dominance operator is not symmetric, $C(A, B)$ is not necessarily equal to $1 - C(B, A)$, and both $C(A, B)$ and $C(B, A)$ must be calculated for understanding how many solutions of A are covered by B and vice versa.

5.2.3 Some Additional Definitions

In this section we present some definitions that are used in our studies for obtaining some of the values presented in the tables and figures or in any of the proposed algorithmic models. Particulary, we define the average of the fitness and the entropy of a population of individuals, and also several mathematical functions as the standard deviation, the median and the interquartile range.

Definition 5.1 (Medium fitness $\overline{f}$). *The medium fitness of the individuals of a population P of size N is defined as the average of the sum of the fitness values of all the individuals composing a population:*

$$\overline{f} = \frac{\sum_{i=1}^{N} f_i}{N} \quad , \tag{5.5}$$

where f_i is the fitness value of individual i. ■

Definition 5.2 (Entropy of the population H_p). *The entropy of a population H_p is calculated as the average of the entropies for each gene of the chromosomes of the individuals:*

$$H_p = \frac{\sum_{i=1}^{L} H_i}{L} \ , \tag{5.6}$$

where H_i is the entropy of the set of values composed by the gene in position i of the chromosome of each individual, with length L. ■

Before defining the standard deviation, we need to know the concept of variance, which is given below:

Definition 5.3 (Variance S). *We define the variance or the quadratic mean deviation of a sample of n observations $X_1, X_2, \ldots X_n$ as:*

$$S = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1} \ , \tag{5.7}$$

where $\overline{X}$ is the arithmetic average of the sample. ■

Hence, the standard deviation can be defined as:

Definition 5.4 (Standard deviation $stdv$). *The standard deviation is defined as the square root of the variance:*

$$stdv = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1}} \ . \tag{5.8}$$

■

When presenting any extreme observation, the average is affected. In this case, it is appropriate to use the median, instead of the mean, for describing the data set:

Definition 5.5 (Median $\widetilde{x}$). *The* median *$\widetilde{x}$ of an ordered set of values $\{x_1, \ldots, x_n\}$ is the value located in the center of an ordered sequence x_i, being $i = (n+1)/2$.* ■

Finally, the definition of the interquartile range is:

Definition 5.6 (Interquartile range IQR). *The interquartile range IQR is a measure of the dispersion of a set of values. It is simply calculated as the difference between the third and the first quartiles:*

$$IQR = Q_3 - Q_1 \ . \tag{5.9}$$

■

5.3 Conclusions

In this chapter we present some new models for improving the behavior of cGAs. Among them, we describe the proposals from which the studied models of the next chapters derive.

Additionally, we show in the second part of the chapter the methodology followed for analyzing the obtained results, and also for comparing our algorithms with other previously existing ones. The reason for the description of this methodology is because a bad planning in the evaluation of the results can lead to inconsistent or wrong results.

6

Design of Self-adaptive cGAs

Intelligence is the ability to adapt to change.

Stephen Hawking (1942 -) – Physicist

As it was concluded in the theoretical study presented in Chap. 4, the use of distinct ratios and update policies in cGAs induces different selection pressures in the algorithm. When extending that study to the empirical field for solving complex problems it was shown that the selection pressure of the algorithms has a marked effect in the search performed by the cGA. Thus, the tradeoff between exploration and exploitation plays an important role in the search performed by the algorithm; furthermore, it is possible to dynamically modify this tradeoff during the search using different strategies. Specifically, we present in this chapter a new model of cGA that dynamically self-adapts the exploration and exploitation capabilities of the search performed. For that, the ratio between population and neighborhood shapes is (automatically) modified for explicitly changing the exploration/exploitation tradeoff during the run. Mainly speaking, this is made according to the evolution of some measure of diversity into the population.

The structure of this chapter is the following one. In the next section we briefly justify the use of adaptive populations in cGAs. The second section (Sect. 6.2) presents some different cGAs characterized by their static, preprogrammed and self-adaptive populations, and they will be analyzed and compared in Sect. 6.3. Finally, the chapter ends with a summary of our main conclusions (Sect. 6.4).

6.1 Introduction

The performance of a cGA may change as a function of several parameters. Among them, we will pay special attention to the ratio in this chapter, which is defined as the relationship between the neighborhood and the population radii, as stated in Chap. 1. Our goal is to study the effects of this ratio on the behavior of the algorithm. As we previously studied in Chap. 4, the use of narrow populations (low ratio values) favors the exploration of the algorithm, while a less narrow population incentives the exploitation of the search space.

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_6, © Springer Science+Business Media, LLC 2008

Since the neighborhood is kept constant in size and shape throughout the studies in this chapter (we always use NEWS), the study of such a ratio is reduced to the analysis of the effects of using different population shapes. The ratio will be smaller as the grid gets thinner.

Reducing the ratio means reducing the global selection intensity on the population, thus promoting exploration. This is expected to allow for a higher diversity that could improve the results in difficult problems (like in multimodal or epistatic tasks). Besides, the search performed inside each neighborhood is guiding the exploitation of the algorithm. We study in this chapter how the ratio affects the search efficiency over a variety of domains. Changing the ratio during the search is a unique feature of cGAs that can be used to shift from exploration to exploitation at a minimum complexity without introducing just another new algorithm family in the literature.

Many techniques for managing the exploration/exploitation tradeoff are possible. Among them, it is worth making a special mention to heterogeneous EAs [4, 20, 127], in which algorithms with different features run in multiple sub-populations and collaborate in order to avoid premature convergence. A different alternative is using *Memetic Algorithms* [184, 199], in which local search is combined with the genetic operators in order to promote local exploitation.

In the case of cGAs, it is very easy to increase population diversity by simply relocating the individuals that compose it by changing the grid shape, as we will see in Sect. 6.2. The reason is that a cellular model provides restrictions on the distance for the mating of solutions due to the use of neighborhoods (one solution may only mate with one of its neighbors). Hence, a cGA can be seen as a mating restriction algorithm based on the Euclidean distance.

6.2 Description of Algorithms

We begin by considering three different static population shapes. Then, we propose two methods for changing the value of the ratio during the execution in order to promote the exploration or exploitation capabilities of the algorithm in certain steps of the search. The first approach to this idea is to modify the value of the ratio at a fixed (predefined) point of the execution. For this goal, we address two different criteria (originally proposed in [26]): (i) changing from exploration to exploitation, and (ii) changing from exploitation to exploration. In the second approach, we propose a dynamic self-manipulation of the ratio value as a function of the evolution progress.

In Fig. 6.1 we show a theoretical idealization of the evolution of the ratio value in the three different classes of algorithms we study in this chapter. We can see how static algorithms (Fig. 6.1a) keep constant the ratio value along the whole evolution, whereas the other two algorithms change it. For pre-programmed algorithms (Fig. 6.1b), this change in the ratio is made in an *a priori* point of the execution, in contrast to adaptive algorithms (Fig. 6.1c), where the ratio varies automatically along the search as a function of the convergence speed of the population.

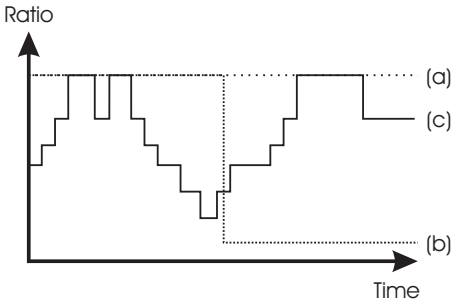


Fig. 6.1. Idealized evolution of the ratio for static (a), pre-programmed (b), and adaptive (c) criteria

Since shifting the ratio is made by changing the shape of the population (and thus its radius) in this work, we theoretically consider the population as a list of length $m \cdot n$, such that the first row of the $m \cdot n$ grid is composed by the n first individuals of the list, the second row is made up with the next n individuals, and so on. Therefore, when performing a change from a $m \cdot n$ grid to a new $m' \cdot n'$ grid (being $m \cdot n = m' \cdot n'$) the individual placed at position (i, j) will be relocated as follows:

$$(i, j) \rightarrow ([i * n + j] \; div \; n', \; [i * n + j] \; mod \; n') \; . \tag{6.1}$$

We call this redistribution method *contiguous*, because the new grid is filled up by following the order of appearance of the individuals in the list. It is shown in Fig. 6.2 how the considered individual plus its neighbors are relocated when the grid shape changes from 8×8 to 4×16, as described in Eq. 6.1. The reader can notice how in general a part of the neighborhood of any given cell is kept while other part may change. In Fig. 6.2 we have drawn the relocation of the individual in position $(2, 4)$ and its neighbors. When the grid shape changes, that individual is relocated at position $(1, 4)$, changing its neighbors placed at its north and south positions, and keeping close those ones placed at its east and west positions. This change in the grid shape can be seen as an actual migration of individuals among neighborhoods, which will introduce additional diversity into the population for the forthcoming generations.

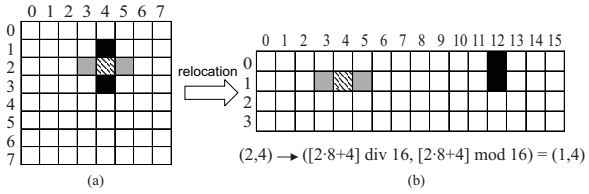


Fig. 6.2. Relocation of an individual and its neighbors when the grid changes from (a) 8×8 to a (b) 4×16 shape

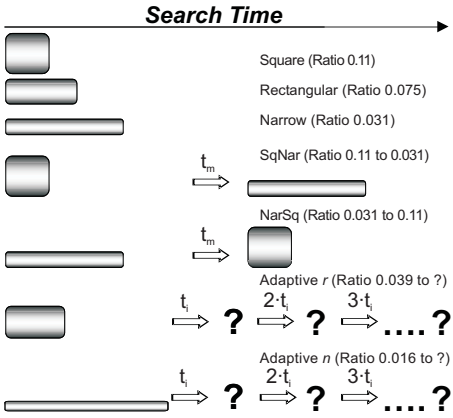


Fig. 6.3. Algorithms studied in this work

All the algorithms studied in this chapter (shown in Fig. 6.3) are obtained from the same canonical cGA by changing only the ratio between the neighborhood and the population topology radii in different manners. We study the influence of this ratio over a representative family of non trivial problems. This family of problems is extended from the initial benchmark included in Chap. 4, where a relationship between low/high ratio and exploration/exploitation of the algorithm is established. In order to help the reader to understand the results, we explain in Sect. 6.2.1 the static and pre-programmed cGAs studied in [26]. After that, we will introduce an adaptive algorithmic proposal with the objective of avoiding the researcher to make an *ad hoc* definition of the ratio, that will (hopefully) improve the efficiency and accuracy of the algorithm (Sect. 6.2.2).

6.2.1 Static and Pre-Programmed Algorithms

In this section we discuss five different algorithms which were initially proposed in [26] and we use in our study. Three of them use static ratios and the other two implement pre-programmed ones (see Fig. 6.3 for more information). Our first aim is to extend this seminal study to a larger and harder set of problems.

First, we tackle a cGA in which three clearly different static ratios have been used (remember we always use NEWS neighborhood):

- *Square*: Ratio = 0.110 (20×20 individuals).
- *Rectangular*: Ratio = 0.075 (10×40 individuals).
- *Narrow*: Ratio = 0.031 (4×100 individuals).

The total population size is always 400 individuals, structured in three different grid shapes, each one with different exploration/exploitation features. Additionally, we have used two more criteria with a dynamic but *a priori* pre-programmed ratio change. These two criteria change the algorithm ratio in a different manner at a fixed point of the execution. In our case, this change is performed in the *middle* of a *typical* execution (t_m). We define the *middle*

of a *typical* run (t_m) as the point wherein the algorithm reaches half the mean number of evaluations needed to solve the problem with the traditional square grid. The pre-programmed algorithms studied change the ratio from 0.110 to 0.031 (we call it *SqNar* –*square* to *narrow*–) and from 0.031 to 0.110 (we call it *NarSq* –*narrow* to *square*–).

6.2.2 Self-Adaptive Algorithms

We present in this section a new contribution to the field of self-adaptation. The core idea, like in most other adaptive algorithms, is to use some feedback mechanism that monitors the evolution and that dynamically rewards or punishes parameters according to their impact on the quality of the solutions. An example of these mechanisms is the method of Davis [60] to adapt operator probabilities in genetic algorithms based on their observed success or failure to yield a fitness improvement. Other examples are the approaches of [28] and [223] to adapt population sizes either by assigning lifetimes to individuals based on their fitness or by having a competition between sub-populations based on the fitness of the best population members.

A main difference here is that the monitoring feedback and the actions undertaken are computationally inexpensive in our case. We really believe that this is the primary feature of any adaptive algorithm in order to be useful for other researchers. The literature contains numerous examples of interesting adaptive algorithms whose feedback control or adaptive actions are so expensive that many other algorithms or hand-tuning operations could result in easier or more efficient algorithms.

Our adaptive mechanisms are shown in Fig. 6.3 (we analyze several mechanisms, not only one). They modify the grid shape during the search as a function of the convergence speed. An important advantage for these adaptive criteria lies in that it is not necessary to set the population shape to any one *ad hoc* value, because a dynamical recalculation of which is the most appropriate one is achieved with a given frequency (t_i) during the search. This also helps in reducing the overhead to a minimum. Algorithm 6.1 describes the basic adaptive pattern; C_1 and C_2 represent the measures of the convergence speed to use.

The adaptive criteria try to increase the local exploitation by changing to the next more square grid shape whenever the algorithm evolves slowly, i.e., when the convergence speed decays below a given threshold value ($\varepsilon \in [0, 1]$). This is achieved changing the shape of the grid to the next one more square, as it is represented by the condition of line 1 in Alg. 6.1. If, on the other side, the search is proceeding too fast, diversity could be lost quickly and there exists a risk of getting stuck in a local optimum. In this second case the population shape will be changed to a narrower grid, thus promoting exploration and diversity in the forthcoming generations. The condition for detecting this situation is expressed by line 3 in Alg. 6.1. If neither C_1 nor C_2 are true, the shape is not changed (line 6).

Algorithm 6.1 Pattern for our dynamic adaptive criteria

```
1. if C1 then
2.     ApproachTo(square)      // exploit
3. else if C2 then
4.     ApproachTo(narrow)      // explore
5. else
6.     Do_not_Change
7. end if
```

As it can be seen, this is a general search pattern, and multiple criteria (C_i) could be used to control whether the algorithm should explore or exploit the individuals for the next t_i generations.

The objective of these adaptive criteria is to maintain the diversity and to pursue the global optimum during the execution. Notice that a change in the grid shape implies the relocation of individuals, which is a kind of migration since individuals in different areas become neighbors. Therefore, this change in the topology is also an additional source for increasing the diversity, intrinsic to the adaptive criterion.

Whenever the criterion is fulfilled, the adaptive search pattern performs a change in the grid shape. The function *ApproachTo*(square/narrow) in Alg. 6.1 changes the grid to the immediately next more square/narrow allowed shape. The bounding cases are the square grid shape and the completely linear (ring-like) shape, as shown in Fig. 6.4. When the current grid is square and the algorithm needs to change to the next "more square" one, or when the present grid shape is the lineal one and the algorithm needs to change to the next "narrower" one, the algorithm does not make any change in the population shape at all, keeping the shape constant. In the rest of cases, when the change is possible, it is accomplished by computing a new position for every individual whose general location is (i, j) as shown in Eq. 6.1. Of course, other changes could have been used, but, in our quest for efficiency, the proposed one is considered to introduce a negligible overhead.

Once the basic adaptive pattern and grid shape modification rules are fixed, we now proceed to explain the criteria used to determine when the population is going "too fast" or "too slow" (criteria C_1 and C_2). We propose in this chapter three different criteria for measuring the search speed. The measures are based on the *average fitness* (criterion AF), the *population en-*

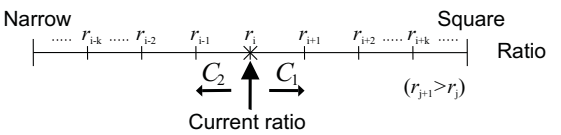


Fig. 6.4. Ratio changes performed when conditions C_1 and C_2 of Alg. 6.1 hold

tropy (criterion PH), or on a combination of they two (criterion $AF{+}PH$). Since these criteria check simple conditions over the average fitness and the population entropy (calculated in every run in the population statistics computation step), we can say that they are inexpensive to measure, and they are indicative of the search states at the same time too. The complexity for calculating the average fitness is $O(\mu)$, while in the case of the population entropy it is $O(\mu \cdot L)$, being μ the size of the population and L the length of the chromosomes. The details on their internals are as follows:

- **AF**: This criterion is based on the *average fitness* of the population. Hence, our measure of the convergence speed is based in terms of the diversity of phenotypes. We define $\Delta\overline{f}_t$ as the difference between the average fitness values in generation t and $t-1$: $\Delta\overline{f}_t = \overline{f}_t - \overline{f}_{t-1}$. The algorithm will change the ratio value to the immediately larger one (keeping constant the population size) if the difference $\Delta\overline{f}_t$ between two contiguous generations (t and $t-1$) decreases at least in a factor of ε: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} < \varepsilon \cdot \Delta\overline{f}_{t-1}$ (condition C_1 of Alg. 6.1). On the contrary, the ratio will be changed to its closest smaller value if that difference increases in a factor greater than $(1-\varepsilon)$: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} > (1-\varepsilon) \cdot \Delta\overline{f}_{t-1}$ (condition C_2). Formally, we define C_1 and C_2 as follows:

$$C_1 ::= \Delta\overline{f}_t < (1+\epsilon) \cdot \Delta\overline{f}_{t-1} \ ,$$
$$C_2 ::= \Delta\overline{f}_t > (2-\epsilon) \cdot \Delta\overline{f}_{t-1} \ .$$

- **PH**: We now propose to measure the convergence speed in terms of the genotypic diversity. The *population entropy* is the metric used for this purpose. We calculate this entropy (H_t) as the average value of the entropy of each gene in the population. Hence, this criterion is similar to AF except for that it uses the change in the population entropy between two generations ($\Delta H_t = H_t - H_{t-1}$) instead of the average fitness variation. Consequently, conditions C_1 and C_2 are expressed as:

$$C_1 ::= \Delta H_t < (1+\epsilon) \cdot \Delta H_{t-1} \ ,$$
$$C_2 ::= \Delta H_t > (2-\epsilon) \cdot \Delta H_{t-1} \ .$$

- **AF+PH**: This is the third and last proposed criterion to create an adaptive cGA. It considers both the population entropy and the average fitness acceleration by combining the two previous criteria (AF and PH). Thus, it relays on the phenotype and genotype diversity in order to obtain the best exploration/exploitation tradeoff. This is a more restrictive criterion with respect to the two previous ones, since although genetic diversity usually implies phenotypic diversity, the reciprocal is not always true. Condition C_1 in this case is the result of the logic `and` operation of conditions C_1 of the two preceding criteria. In the same way, C_2 will be the `and` operation of conditions C_2 of AF and PH:

$$C_1 ::= (\Delta\overline{f}_t < (1+\epsilon)\cdot\Delta\overline{f}_{t-1}) \textbf{ and } (\Delta H_t < (1+\epsilon)\cdot\Delta H_{t-1}) \ ,$$
$$C_2 ::= (\Delta\overline{f}_t > (2-\epsilon)\cdot\Delta\overline{f}_{t-1}) \textbf{ and } (\Delta H_t > (2-\epsilon)\cdot\Delta H_{t-1}) \ .$$

Moreover, for each adaptive criterion (AF, PH, and AF+PH) we can start the execution by using (1) the square grid shape, (2) the narrowest one, or (3) the one having a medium ratio value (rectangular). In our terminology, we will specify the initial grid shape by adding at the beginning of the criterion name a *s*, *n*, or *r* for executions that begins at the square, narrowest, or rectangular ratio grid shapes (the one with the average ratio value), respectively. For example, the algorithm which uses criterion AF starting with the narrowest shape will be denoted as *n*AF.

In short, we have proposed in this subsection three different adaptive criteria. The first one, AF, is based on the phenotypic diversity, while the second one, PH, is based on genotypic diversity. Finally, AF+PH is a combined criterion accounting for diversity at these two levels.

6.3 Experimentation

We present here the comparison of the algorithms presented in the previous section. For our test, we have selected a representative benchmark because it contains problems with many different interesting features in optimization, like epistasis, multimodality, deceptiveness, use of constraints, parameter identification, and problem generators.

Initially, the studied instances are the same ones of the benchmark used in Chap. 4, extended with some more instances of problems MAXCUT (instances MAXCUT20.01, MAXCUT20.09 and MAXCUT100), and MTTP –instances MTTP20, MTTP100 and MTTP200– (see Appendix A for more information on these problems). The election of this benchmark is justified for both the difficulty of the problems composing it and also the application domains they belong to (combinatorial optimization, continuous optimization, telecommunications, planning, etc.). This guarantees a high level of confidence in the results, although the evaluation of the conclusions will be a more laborious task than in the case of using a small number of problems.

Although a full-length study of the problems presented in the previous section is beyond the scope of this work, in this section we present and analyze the results obtained when solving all the problems with all the discussed cGA variants (statics, pre-programmed and dynamics), always with a constant neighborhood shape (NEWS). Note that it is not our aim here to compare cGAs performance with state-of-the-art algorithms and heuristics for combinatorial optimization. To this end, we should at least do a previous study for tuning the parameters or include local search capabilities in the algorithm, which is not the case. Thus, the results are only used for the comparison of the relative performance of the different proposed self-adaptive cGAs.

The resulting work is a direct continuation of a past one [26], extended here by including more problems in the test suite and some new algorithms (the self-adaptive cGAs). We proceed in three stages, namely, analysis of static ratios, dynamic pre-programmed ratios, and dynamic adaptive ratios. In the two first stages we use square and rectangular grids in *a priori* well-known points of the evolution, i.e., either using always the same grid shape (static), or setting a fixed change in the grid shape before the optimization.

In the third stage we apply three (dynamic) mechanisms for self-adapting the grid shape based on the convergence speed, as explained in Sect. 6.2.2. We study these mechanisms with two different initial values for the ratio: the smallest one, and that with the middle value (i.e., the median of the allowed set of values for the ratio).

All the algorithms have been compared in terms of the efficiency and efficacy, so Tables 6.2 and 6.3 show the average number of evaluations needed to solve each problem with all the algorithms, and the percentage of successful runs after making 100 independent runs for each problem. We have performed statistical tests on the results in order to obtain statistically significant conclusions in our comparisons (see Tables 6.2 and 6.3).

We have organized this section into three subsections. In the first one we describe the parameters used for the execution of the algorithms. In the next subsection we present and analyze the results obtained with the different algorithms for all the problems. Finally, we offer an additional global graphic interpretation of the results in the last subsection.

6.3.1 Parameterization

In order to make a meaningful comparison among all the algorithms, we have used a common parameterization. The details are described in Table 6.1, where L is the length of the string representing the chromosome of the individuals. Regarding the selection method used, one parent is always the individual itself, while the other one is obtained by using *Binary Tournament* –BT–. The two parents are forced to be different. These two parents are recombined using DPX. From the two offsprings obtained by DPX, we only consider one of them: the one having the largest portion of the best parent. The recombination is always applied (probability $p_c = 1.0$). The bit mutation probability is set to $p_m = 1/L$. The exceptions are COUNTSAT, where we use $p_m = 1/(2 \cdot L)$, and the FMS problem, for which a value of $p_m = 10/L$ is used. These two probability values are needed because the algorithms had a negligible solution rate with the standard $p_m = 1/L$ probability for these two problems.

We will replace the considered individual on each generation only if its offspring has a better fitness value. This replacement strategy is called *replace if better* [231]. The cost of solving a problem is analyzed by measuring the number of evaluations of the objective function made by the algorithm during the search. Finally, the stop condition for all the algorithms is to find a solution or to achieve a maximum of one million function evaluations.

Table 6.1. Parameterization used in our algorithms

Population size	400 individuals
Parent selection	Current + binary tournament
Recombination	DPX, $p_c = 1.0$
Bit mutation	Bit-flip, $p_m = 1/L$
Replacement	Rep_if_Better
Stop condition	Find a solution or reach 10^6 evals

As we previously introduced in Chap. 5, we performed a statistical study on the obtained solutions in order to assess statistical significance on the results, and we consider a 0.05 level of significance. Statistical significative differences among the algorithms are shown with symbol '+' in Tables 6.2 to 6.4, while non-significance is shown with '•'.

In short, we have eleven different algorithms with the parameterization shown in Table 6.1. Three of the studied algorithms use static ratios, two others employ dynamic *pre-programmed* ratios, and the last six ones make use of the proposed *dynamic adaptive* criteria.

6.3.2 Experimental Results

In this section we present the results obtained when solving the problems of our benchmark with all the proposed algorithms. The *globally* best values throughout this chapter for each problem are **bolded**; for example, the best tested algorithm that solves the FMS problem with the lowest effort is called *SqNar* (pre-programmed criterion), as shown in Table 6.2.

This subsection is actually organized into two parts. In the first part, a study of the results obtained with the different static and pre-programmed algorithms is developed for all the problems. The second part contains this same study, but in this case for the adaptive algorithms.

Static and Pre-programmed Ratios

In this section we tackle the optimization of all the problems employing static and pre-programmed ratios. Our results confirm those of [26] in which narrow grids were quite efficient for multimodal and/or epistatic problems, while the square and rectangular grids performed better in the case of simple and non epistatic problems. It must be taken into account that here we are using the BT selection method, while in [26] RW was used. The use of BT selection is expected to reduce the sampling error in small neighborhoods with respect to fitness proportionate selections.

Looking at Table 6.2 we can see that narrow grids are suitable for multimodal and epistatic problems (*narrow* obtains better results than the others for P-PEAKS, and *SqNar* obtains the best results for FMS). Conversely, these narrow grids perform worse than the others in the case of MMDP (deceptive), and some instances of MTTP (combinatorial optimization). Finally, the two pre-programmed criteria have the worst performances in the case of ECC. In the other problems there are no statistically significant differences.

Table 6.2. Results with static and pre-programmed ratios

Problem	Static			Pre-programmed		Test
	Square	Rectangular	Narrow	SqNar	NarSq	
MMDP	160090.30	182305.67	247841.13	148505.48	172509.28	+
	93%	**96%**	93%	86%	92%	
FMS	435697.94	494400.25	499031.79	**355209.39**	462288.25	+
	59%	67%	68%	42%	59%	
P-PEAKS	54907.93	54984.12	50853.82	63545.47	60674.31	+
	100%	**100%**	**100%**	**100%**	**100%**	
COUNTSAT	6845.56	6851.26	6856.77	6845.18	6849.17	•
	24%	54%	83%	22%	43%	
ECC	167701.21	169469.62	156541.38	190875.00	187658.98	+
	100%	**100%**	**100%**	**100%**	**100%**	
MAXCUT20.01	6054.10	5769.39	5713.25	5929.79	5785.43	•
	100%	**100%**	**100%**	**100%**	**100%**	
MAXCUT20.09	**9282.15**	9606.96	9899.69	9775.38	10260.59	•
	100%	**100%**	**100%**	**100%**	**100%**	
MAXCUT100	263616.77	217509.87	197977.18	223909.83	240346.84	•
	63%	55%	53%	55%	53%	
MTTP20	6058.11	5753.35	5681.17	6134.30	**5584.93**	+
	100%	**100%**	**100%**	**100%**	**100%**	
MTTP100	194656.43	201409.27	206177.16	190566.23	194371.72	+
	100%	**100%**	**100%**	**100%**	**100%**	
MTTP200	565364.89	566279.17	606146.59	565465.14	570016.49	+
	100%	**100%**	**100%**	**100%**	**100%**	

Although it is not a globally valid strategy for any unseen problem, our conclusions up to now do explain the common belief that a *good* optimization algorithm must initially seek promising regions and then gradually search in the neighborhood of the best so far points. In this process, there is a clear need of avoiding full convergence to a local optimum, which exactly describes the behavior of the algorithm *SqNar*, since it shifts to a narrow grid to avoid such premature convergence in the second half of the search. In effect, this algorithm has a high accuracy and efficiency on this benchmark.

In summary, these results also conform to the No Free Lunch Theorem (NFL) [265], which predicts that it is not possible to find "a globally best" algorithm for all possible problems. Thus, we included in the test suite a large number of different problems, since only analyzing two or three problems can lead to an undesirable bias in the conclusions. Each algorithm has then been pointed out as the most efficient for a different subclass of problems. However, by inspecting Table 6.2, we can observe just three bolded results (statistically significant differences only for FMS, in general), what means that the overall most efficient algorithm has still to be discovered in this work. As we will see, the different adaptive criteria evaluated in the next subsection will be able of providing the optimum solution at a lower cost with respect to all the non adaptive ratios for every tested problem (with the exception of FMS).

Adaptive Criteria

In this section we continue our work by addressing the study of the adaptive algorithms proposed in Sect. 6.2.2. For each adaptive criterion, we have made two different tests: one beginning at the grid shape with a middle ratio value (r), and the other starting at the lowest one (n).

Table 6.3. Results with the adaptive criteria

Problem	nAF	rAF	nPH	rPH	nAF+PH	rAF+PH	Test
MMDP	155342.46	**137674.50**	180601.45	160747.96	144483.41	162039.13	+
	93%	83%	93%	91%	90%	**96%**	
FMS	493449.91	491922.13	528850.10	555155.67	465396.00	439310.97	+
	64%	61%	73%	**76%**	59%	57%	
P-PEAKS	56475.84	58031.72	**49253.83**	49855.33	53127.49	54350.54	+
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
COUNTSAT	5023.71	**4678.77**	5296.32	5188.08	4999.59	5252.22	•
	82%	**90%**	89%	86%	88%	88%	
ECC	178921.19	179081.59	**151256.20**	156008.05	193373.23	184058.00	+
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
MAXCUT20.01	5416.51	5641.07	5701.22	6082.17	**5340.32**	5657.11	•
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
MAXCUT20.09	9554.83	9659.09	9430.52	10015.98	9935.78	9598.94	•
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
MAXCUT100	131359.13	142470.83	151048.22	**111662.07**	132574.18	141945.57	•
	45%	46%	46%	39%	43%	41%	
MTTP20	5729.29	5941.82	6206.48	5941.82	6038.06	5853.60	•
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
MTTP100	190425.88	192179.23	**190081.02**	196958.17	192871.98	192579.25	•
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	
MTTP200	529158.60	523913.52	**523372.17**	525349.10	531019.24	541397.12	•
	100%	**100%**	**100%**	**100%**	**100%**	**100%**	

Also, since the adaptive criteria rely on a small ε value to define what is meant by "too fast convergence" we have developed a first set of the tests for all the criteria with different ε values: 0.05, 0.15, 0.25, 0.3. From all this plethora of tests we have finally selected $\varepsilon = 0.05$ as the best one. In order to get this conclusion two criteria have been applied, and in both of them $\varepsilon = 0.05$ was the winner. Of course, values other than $\varepsilon = 0.05$ could reduce the effort for a given problem, but we need to select a given value to focus our discussion. The used selection criteria look for an ε value which allows us to obtain (1) the larger accuracy, and (2) the best efficiency. We wanted to make such a formal and statistical study to avoid an *ad hoc* definition of ε, but we do not provide the details in this chapter since including the results, tables, and explanations could have distracted the attention of the reader. The reader interested in those details can see [12].

Next, we will discuss the results we have obtained with our adaptive cGAs (see Table 6.3), and we will also compare our best adaptive algorithms with all the studied static and pre-programmed ones for each problem. There are statistically significant differences among the adaptive criteria of Table 6.3 in just some few cases. We will just emphasize the highly desirable behavior of nPH and rPH, which are (with statistical significance) the best adaptive criteria for P-PEAKS and ECC (and slightly worse than rAF and nAF+PH for the FMS problem).

Comparing the adaptive criteria with the cGAs using static and pre-programmed ratios, we can see (Tables 6.2 and 6.3) that the adaptive algorithms are more efficient in general than the static and pre-programmed ones, standing out the nPH algorithm over the others.

Table 6.4. Comparison of the best adaptive criterion versus non adaptive ratios

Problem	Best Adaptive Criterion	Square	Rectangular	Narrow	SqNar	NarSq
MMDP	rAF	•	•	+	•	•
FMS	rAF+PH	•	•	•	•	•
P-PEAKS	nPH	+	+	•	+	+
COUNTSAT	rAF	+	•	•	•	•
ECC	nPH	+	+	•	+	+
MAXCUT20.01	nAF+PH	•	•	•	•	•
MAXCUT20.09	nPH	•	•	•	•	•
MAXCUT100	rPH	•	•	•	•	+
MTTP20	nAF	•	•	•	•	•
MTTP100	nPH	•	•	+	•	•
MTTP200	nPH	+	+	+	+	+

In terms of the efficacy of the algorithms (success or hit rate), the adaptive algorithms find, in general, the optimal value more frequently (after 100 independent runs) than the static and pre-programmed ones. Moreover, these adaptive algorithms obtain the best hit rates for almost every problem.

In Table 6.4 we compare, for each problem, our best adaptive algorithm versus the studied static and pre-programmed ones. Symbol '+' means that the adaptive algorithm is better with statistical confidence than the non adaptive ones compared, while non-significance is shown with '•'. We can see that there is not any static or pre-programmed algorithm better than the best adaptive one for each problem. Hence, the bolded results corresponding to a better non adaptive algorithm in Table 6.2 are not statistically different with respect to the adaptive ones. For all problems, the existing statistical differences promote the adaptive criteria versus the static and pre-programmed ones.

6.3.3 Additional Discussion

If we seek among all the adaptive criteria we then conclude (as expected from the theory) that there is no adaptive criterion that significatively outperforms the rest for all the test suite, according to the statistic tests performed. However, it is also clear after these results that adaptive ratios are the first kind of algorithm one must select in order to have a really good baseline for comparisons. For the problems evaluated, and for other problems sharing their same features, we make a contribution by defining new competitive and robust algorithms that can improve the existing performance at a minimum implementation requirement (changing the grid shape).

In order to provide an intuitive explanation of what is happening during the search of the algorithms we first track the evolution of the ratio value along the search. Secondly, we have taken some snapshots of the population at different points of the evolution in order to study the diffusion of solutions through the population. Finally, we have studied a representative convergence scenario of an adaptive algorithm through the evolution.

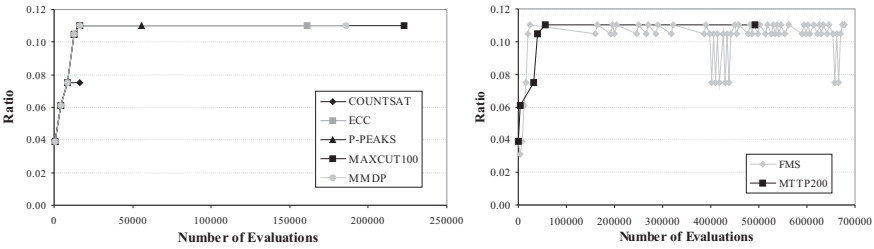


Fig. 6.5. Dynamics of the ratio observed when using rAF

In Fig. 6.5 we have drawn an example of the ratio fluctuation automatically produced by the rAF adaptive criterion during a typical execution on all the problems (we just plot the hardest instances in the case of MAXCUT and MTTP: MAXCUT100 and MTTP200). We have plotted separately in rigth part of the figure the hardest problems: MTTP with 200 variables – MTTP200– and FMS.

It can be noticed in Fig. 6.5 a clear trend towards promoting exploitation of the population (ratios over 0.1). This general trend to evolve towards the square shape is expected, because the search focuses towards population exploitation after an initial phase of exploration. This shift from small to high ratios is detected in all the problems. However, the adaptive algorithms seem to need a more complex search of the optimal ratio when facing the most difficult problems: FMS and MTTP. In these two cases (right-hand plot of Fig. 6.5) there are brief and periodic visits to smaller ratios (below 0.08) with the goal of improving the exploration of the algorithm. This automatic alternate shifting between exploitation and exploration in some problems is a noticeable feature of the adaptive mechanism we propose, because it shows how the algorithm decides by itself when to explore and when to exploit the population.

But we want to show that this search for the appropriate exploration/exploitation tradeoff can be achieved in a gradual and progressive way. In order to prove this claim we zoom into the behavior of the nAF+PH algorithm for the FMS problem (Fig. 6.6). We notice that when the algorithm finds difficulties during an exploiting phase (high ratio), it automatically decreases the ratio (even repeatedly) to improve the exploration. This provides, in general, a higher hit rate for the adaptive cGAs with respect to the other studied cGAs (statics and pre-programmed). In Fig. 6.6 we can see the drop of the ratio value is performed by the adaptive algorithm to appropriately regulate by itself the speed up of the search for the optimum.

Now, let us turn to analyze the evolution for MAXCUT and FMS from a phenotypic diversity point of view. The pictures in Figs. 6.7 and 6.8 have been taken at the beginning, middle, and end of a typical execution with an adaptive criteria which begins using the narrowest population shape (specifically nAF+PH). These figures show three snapshot representations of the fitness distribution in the population at different stages of the search. Different gray

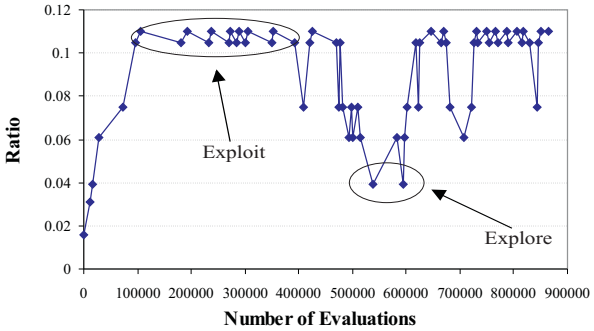


Fig. 6.6. Evolution of the ratio for the FMS problem with algorithm nAF+PH

tones mean different fitness values, and darker gray tones correspond to the best individuals. It can be observed in the two figures that diversity (variety of gray tones) decreases during the run, as expected.

The particular case of MAXCUT100 (Fig. 6.7) is a sample scenario of a fast evolution towards the square population shape during the execution; once the algorithm has reached the square shape, it is maintained until the solution is found (see also Fig. 6.5). The reason for this behavior is that the exploration/exploitation tradeoff that the algorithm finds is adequate to maintain a permanent and not too fast convergence of the individuals towards the solution. This convergence can be observed in Fig. 6.7, where we can notice that the number of grey tones in the population decreases along the execution. The convergence is noticeable at the middle of the execution (Fig. 6.7b) and continues growing until the optimal solution is found (Fig. 6.7c). Also, since the same ratio value is kept during all the evolution (except at the beginning), no relocation of individuals (which introduces diversity) is made, and the algorithm inherently creates islands of similar fitness values, also called *niches* (Fig. 6.7c), corresponding to those areas with the same gray tone. These different areas represent distinct convergence paths motivated by the smooth diffusion of good solutions, a distinctive feature of cGAs. In particular, the exploration of the search space is specially enforced in the bounds of these niches.

Fig. 6.7. Population at the beginning (a), middle (b), and end (c) of a sample execution of MAXCUT (instance MAXCUT100) with nAF+PH

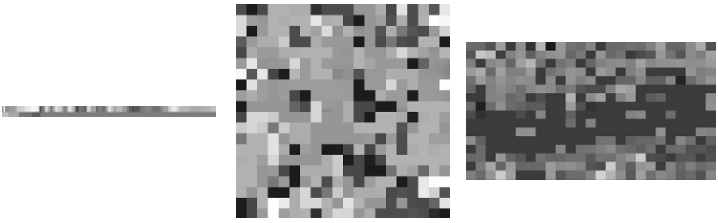


Fig. 6.8. Population at the beginning (a), middle (b), and end (c) of a sample execution of FMS with nAF+PH

In the case of FMS, a similar scenario appears; in this case we plot an example situation in which the algorithm is near to converge to a local optimum. As soon as the adaptive criterion detects such condition it tries to get out of this local optimum by promoting exploration (as it also occurs in Fig. 6.6). We can see in Fig. 6.8b that the diversity is smaller than in the case of MAXCUT in the middle of the execution. Since most individuals have similar fitness values, when a better solution appears in the grid it will be spread through all the population reasonably fast. This effect produces an acceleration in the mean fitness of the individuals and, therefore, a change to the next more rectangular grid shape (if possible) is performed, thus introducing some more diversity in the neighborhoods for the next generations. Contrary to this reasoning, a high level of diversity is maintained during all the evolution because, due to the epistatic nature of the problem, small changes in the individuals lead to big differences in the fitness values. That is the reason, as well as the high number of ratio changes made for this problem, for the absence of homogeneous colored areas. It also justifies the important levels of diversity present at the end of the evolution for FMS (in relation to the MAXCUT problem).

Finally, we want to illustrate the differences in the convergence speed of the three main classes of the studied algorithms (static, pre-programmed, and adaptive). Hence, we plot in Fig. 6.9 the maximum fitness value in every generation in typical runs for three different algorithms (*Square*, *SqNar*, and rPH) when solving the instance of 100 vertices of the MAXCUT problem (MAXCUT100). As it can be seen, the evolution of the best so far solution during the run is quite similar in the three algorithms. The difference in the behavior of the algorithms can be seen at the end of the evolution, where *Square* and *SqNar* get stuck during many generations before finding the optimal solution, while the adaptive algorithm, due to the diversity of the population, finds quickly an optimum.

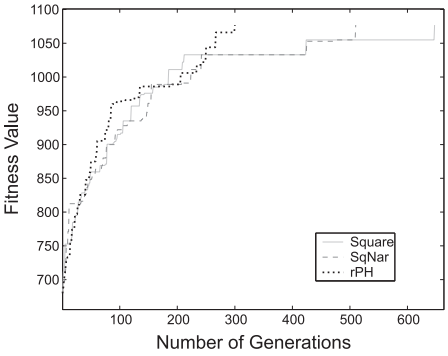


Fig. 6.9. Evolution of the maximum fitness found with MAXCUT100

6.4 Conclusions

In this work we have analyzed the behavior of many cellular GAs over an assorted set of problems. The ratio, defined as the relationship between the neighborhood and the population radii, represents a unique parameter rich in possibilities for developing efficient algorithms based on the canonical cGA. As the population is distributed in a 2D toroidal grid, the studied models are all embedded in many other similar algorithms; thus, we hope these results to be broadly helpful for other researchers.

Our main motivation for this work has been to advance in the knowledge of the appropriate exploration/exploitation tradeoff for an algorithm. Specifically, we have studied the influence of the ratio on the field of dynamical structured populations. This feature is, to our knowledge, used in this work for the first time.

Technically, we can confirm the results in [26], which concluded that a narrow grid (small ratio) is quite efficient for multimodal and/or epistatic problems, while it performs worse with respect to the square and rectangular grids (high ratios) in the case of simple and non epistatic problems. Additionally, we have demonstrated the importance of using an intermediate grid (*rectangular*) for combinatorial optimization problems in this work.

We must emphasize that any of the developed adaptive criteria is just a seed idea on how to obtain competent search techniques. An important feeling after this work is that the search can be easily guided by using structured populations. These kind of models are very efficient from a numerical point of view, while they still admit any advanced technique developed in the EA field to be included in their basic execution. These adaptive algorithms outperform the other studied ones for all the proposed problems, and the exceptions do not have statistical differences.

7

Design of Cellular Memetic Algorithms

Almost everybody that's well-known gets tagged with a nickname.

Alan Alda (1936 -) – Actor

Memetic algorithms (MAs) [154, 184] is a recent nickname for techniques combining features of different metaheuristics (also exact methods), such as population based algorithms (EAs) and local search procedures; restart techniques and intensive hybridization with problem knowledge is also supposed to be the ingredients of a MA. The salient feature of MAs is that the tradeoff between exploration and exploitation they perform on the search space is addressed explicitly in most approaches by using the longstanding concepts of hybrid optimizers [61].

In this chapter, we intend to take profit from the advantages reported by MAs when used in combination with cGAs. For that, a new model of highly specialized cGA is designed here to guide the reader on how cellular memetic algorithms can be constructed. The basic idea is to introduce the MA concepts into a structured population of tentative solutions much in the style of cGAs. For evaluating our algorithms we have selected the satisfiability problem (SAT), described in Appendix A. This is a hard combinatorial problem, well-known in the literature, and having important practical applications, such as planning [143] and image interpretation [213], among others. Besides its importance in research, we selected SAT since readers are most probably familiar with it, and thus they can concentrate on the algorithm proposal itself.

In the field of evolutionary computation, the latest advances found in literature clearly show that EAs can yield good results for SAT when hybridizing them with additional techniques, e.g., adaptive fitness functions, problem-specific operators, or local optimization [32, 64, 83, 91, 120].

The motivation for this chapter is to study the behavior of different cMAs having specific fitness functions and embedding some knowledge on the problem into the recombination, mutation, and local search operators, as well as into the problem representation.

We will compare these new proposals against the basic local search heuristics working alone, and versus two canonical cGAs (without any local search). Additionally, two different ways of embedding local search have been analyzed:

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_7, © Springer Science+Business Media, LLC 2008

(i) a computationally light local search step applied to every individual, or (ii) an in depth exploitation local search step applied with a low probability. The numerical work is an extension to [15, 16], including a preliminary study justifying the algorithms used plus a comparison of our results versus those of other well-known algorithms in the literature (a must in effective research).

This chapter is organized in the following manner. In Sect. 7.1, our new proposal for a cMA is presented. Section 7.2 introduces all the algorithms involved in our analysis, including three simple methods (GRAD, Simulated Annealing –SA– and WSAT), and a basic cGA. Later, several cMAs will be presented, which are the result of different combinations of a basic cGA with the previous component algorithms. The experimentation and its discussion are summarized in Sect. 7.3, while the conclusions and future research directions are addressed in Sect. 7.4.

7.1 Cellular Memetic Algorithms

Memetic algorithms [154, 184] are search algorithms in which knowledge of the problem is used in operators or other algorithmic components with the objective of improving the behavior of the original algorithm. Not only local search, but also restart techniques, and structured or intensive search are commonly considered in MAs [184]. All these ideas are well-known because they lead to hybridization, a domain very popular in EAs from its beginning. Thus, the way to start is clear: let us design a basic cellular memetic algorithm (cMA) to illustrate the concepts.

In Alg. 7.1 we show a first pseudo-code for a canonical cMA. As it can be seen, the main difference between the pseudo-codes of the canonical cMA and cGA is the local search step included (line 8 in Alg. 7.1). Besides, some

Algorithm 7.1 Pseudo-code of a canonical cMA

```
1.  cMA(cma)        //Algorithm parameters in 'cma'
2.  GenerateInitialPopulation(cma.pop);
3.  Evaluation(cma.pop);
4.  while !StopCondition() do
5.    for individual ← 1 to cma.popSize do
6.      neighbors ← GetNeighbors(cma,position(individual));
7.      parents ← Selection(neighbors);
8.      offspring ← Recombination(cma.P_c, parents);
9.      offspring ← Mutation(cma.P_m, offspring);
10.     offspring ← LocalSearch(cma.P_LS, offspring, {intensive|light});
11.     Evaluation(offspring);
12.     Insert(position(individual), offspring, cma, auxPop);
13.   end for
14.   cma.pop ← auxPop;
15. end while
```

knowledge of the problem can be included in different parts of the cMA, like the variation operators (lines 7 to 10), the evaluation of the fitness function (lines 3 and 11), the process for generating the initial population (line 2), or the problem representation.

7.2 Simple and Advanced Components in Cellular MAs

Since we are facing the simplest way of constructing a memetic algorithm (namely hybridization with local search) let us first present several algorithms that can be used as local search optimizers inside a cGA. The reader can find such a description in the current section organized in the form of three subsections. Specifically, we first study three simple methods for solving SAT, then a basic cGA –implemented in JCell–, and finally the proposed cMAs.

7.2.1 Three Basic Local Search Techniques for SAT

In this subsection, we present the three local search procedures (LS) that later will be used to built our cMAs; these three methods can of course be independently used for solving SAT. Two of them were specifically designed for this problem: (i) a gradient algorithm (GRAD), based on the flip heuristic, and especially developed for this work by the authors, and (ii) the well-known WSAT algorithm. The third procedure is Simulated Annealing (SA), a well-known general purpose metaheuristic.

GRAD

For this work, we have developed a new local search algorithm (called GRAD) for SAT. GRAD is an algorithm based on the flip heuristic [120] which performs a greedy search in the space of solutions (see Alg. 7.2). Basically, GRAD resorts to mutations of the value of a variable according to the number of non-satisfied clauses that the variable belongs to: the higher the number of unsatisfied clauses a variable belongs to, the higher the probability for mutating (flipping) it. As it can be seen in Alg. 7.2, some noise is added to the search (with probability 0.2) in order to enhance its exploration capabilities. The main difference of GRAD with respect to the pre-existing flip heuristic is that the latter flips a variable (v) in terms of the gain of that flip: $v = \left\{ v_i / \max(\text{sat\_clauses}(\overline{\text{valueof}(v_i)}) - \text{sat\_clauses}(\text{valueof}(v_i))) \right\}$ (with $i = 1$ to the number of variables, and sat_clauses(valueof(v_i)) the number of satisfied clauses containing the value of v_i), while in GRAD the flip is made to every variable (v) satisfying $v = \{v_i / \max(\text{unsat\_clauses}(\text{valueof}(v_i)))\}$ with the same probability (independently of the gain). The expression unsat_clauses(valueof(v_i)) is the number of non-satisfied clauses containing the value of v_i. This difference makes GRAD computationally lighter than the flip heuristic and thus more suited to be embedded into a higher level algorithm like a cGA.

Algorithm 7.2 Pseudo-code of GRAD

```
 1. GRAD(problem)
 2. best_ind = NewRandomIndividual();
 3. while !StopCondition() do
 4.    ind = NewRandomIndividual();
 5.    for steps ← 0 to MAX_STEPS do
 6.       if rand0to1() < noise_prob then
 7.          Mutate(ind,RandomVariableIntoAnyUnsatisfiedClause());
 8.       else
 9.          vars_to_flip[]=VarsInvolvedInTheMaximumNumberOfNonSatisfiedClauses();
10.          MutateWithEqualProbability(ind,vars_to_flip);
11.       end if
12.       Evaluation(ind);
13.       best_ind = Best(ind,best_ind);
14.    end for
15. end while
```

The detailed functioning of GRAD is simple. The algorithm starts by randomly generating both and initial best solution and the first individual (lines 2 and 4, respectively). After that, and until the final condition is met, the algorithm repeatedly generates a new individual (offspring) from the current one (parent), evaluates it, and replaces the best current solution with it if it is better (higher fitness value, since we are maximizing). The offspring is created by flipping the worst genes of the parent –those unsatisfying the largest number of clauses– with equal probability (lines 9 and 10). With a preset probability, some noise (20%) is introduced in the search to help escaping from local optima. In this case, the offspring is mutated by flipping the value of a randomly chosen variable of the parent unsatisfying one or more clauses (lines 6 and 7). Then, the search process is repeated for the offspring (lines 5 to 14). Every `MAX_STEPS` iterations the search is restarted –i.e., the current individual is randomly generated– (line 4). We have set this value to 10 times the number of variables in practice. The algorithm stops (line 3) when the optimal or the best-known solution is found or after making 2 millions of fitness function evaluations (a true top bound for the effort in most problems).

WSAT

The WSAT algorithm [227] is a greedy heuristic specifically designed for SAT. Basically, it consists in repeatedly selecting a non satisfied clause (uniformly random) and flipping one of its variables (see Alg. 7.3). There exist several methods for selecting this variable [179]. Among them, we have adopted the BEST strategy, which consists in flipping a variable of the clause with a given probability (prob_noise = 0.5), and otherwise flips the variable that minimizes the number of clauses that are true in the current state, but that would become false if the flip were made. After a number of steps (line 3), the search is

Algorithm 7.3 Pseudo-code of WSAT

```
1.  WSAT(problem)
2.  best_ind = NewRandomIndividual();
3.  while !StopCondition() do
4.     ind = NewRandomIndividual();
5.     for steps ← 0 to MAX_STEPS do
6.        clause = RandomNonSatisfiedClause()
7.        if rand0to1() < noise_prob then
8.           Mutate(ind,clause[randomInt(clause_length)]);
9.        else
10.          for i ← 0 to clause_length do
11.             lost_clauses[i] = LostClausesAfterFlip(i);
12.          end for
13.          Mutate(ind, clause[IndexOfMinValue(lost_clauses)]);
14.       end if
15.       Evaluation(ind);
16.       best_ind = Best(ind, best_ind);
17.    end for
18. end while
```

"restarted" by replacing the current individual by a randomly generated one (line 4). Like in the case of GRAD, we "restart" every 10 times the number of variables steps, and the best found solution so far is always tracked.

SA

Simulated Annealing [145] is probably one of the first metaheuristics with an explicit strategy to escape from local optima (see Alg. 7.4 for a pseudo-code). The core idea is to sparsely allow for movements resulting in solutions of worse quality in order to escape from local optima. For that, a parameter called temperature (`Temp`) is used, such that the probability of accepting a worse individual is:

$$p(\text{Temp, offspr, ind}) = e^{\frac{(\text{Get\_Fit(offspr)} - \text{Get\_Fit(ind)}) \cdot \mathbf{10^4}}{\text{targetFitness} \cdot \text{Temp}}} \quad . \tag{7.1}$$

The value of this probability decreases during the execution (line 19) in order to reduce the probability for accepting movements with a decrement in the quality of the solution (computed as shown in Eq. 7.1) along the search. `Temp` is initialized to a given upper bound $\mathtt{T_{max}}$ (line 5), and new individuals are computed while the current value of `Temp` is larger than a given threshold $\mathtt{T_{min}}$ (lines 6 to 20). The value of the temperature decreases in terms of the parameter `coolingRate`. If the new individual is better (higher fitness value) than the best so far one it is accepted as the new best one (lines 14 and 15) or, otherwise, it replaces the best one with a given probability (lines 16 and 17). After some experimental tests, we have set these values to $\mathtt{T_{max}} = 10$, $\mathtt{T_{min}} = 1$, and $\mathtt{coolingRate} = 0.8$.

Algorithm 7.4 Pseudo-code of SA

```
1. Simulated_Annealing(problem, T_max, coolingRate)
2. ind = NewRandomIndividual();
3. best_ind = ind;
4. while !StopCondition() do
5.    Temp = T_max;
6.    while Temp > T_min do
7.       offspring = ind;            // generate an offspring
8.       for i ← 0 to problem.num_vars do
9.          if rand0to1() < 1/problem.num_vars then
10.            Mutate(offspring, i);
11.         end if
12.      end for
13.      Evaluate(offspring);
14.      if Fitness(offspring) >= Fitness(ind) then
15.         ind = offspring;
16.      else if rand0to1() < p(Temp, offspring, ind) then
17.         ind = offspring;
18.      end if
19.      Temp *= coolingRate;
20.   end while
21.   best_ind = Best(offspring, best_ind);
22. end while
```

7.2.2 Cellular Memetic GAs

Let us address the design of cMAs here for SAT. We will hybridize a basic cGA (implemented in JCell) with other component; a component in the resulting cMA could be recombination or mutation operators (either general or SAT-oriented), and local search (see Table 7.1). In addition, we will use an adaptive fitness function specially designed for SAT: SAW (whose details could be found in Appendix A). All these components could be used separately or combined in a given cMA.

We use two specific genetic operators for recombination and mutation which are called *Unsatisfied Clauses Recombination* (UCR) and *Unsatisfied Clauses Mutation* (UCM), respectively. These two operators focus on keeping constant the values of the variables satisfying all the clauses they belong to. Our UCR is exactly the same operator as the proposed C_B in [121], and UCM

Table 7.1. Operators used in JCell in this work for solving SAT

Operator	Generic	Specific
Crossover	DPX	UCR
Mutation	BM	UCM
Local Search	SA	GRAD
		WSAT

Algorithm 7.5 Pseudo-code for UCM

```
1. UCM(Indiv, Noise)
2. if rand0to1() ≤ Noise then
3.     Flip(Indiv, randomInt(Indiv.length));
4. else
5.     M_B(Indiv);
6. end if
```

is the result of adding some noise to M_B (previously proposed in [121]), as seen in Alg. 7.5. On the other hand, we use two well-known generic recombination and mutation operators: two point recombination (DPX) and binary mutation (BM).

As to the other three heuristic methods proposed hereinbefore, they have been adopted as LS operators in our cMA. As it can be seen in Sect. 7.3, some different configurations of these local search methods have been studied. These configurations differ on the probability of applying the local search operator to the individuals (frequency of utilization) and the intensity of the local search step once it is applied. The idea is to regulate the overall computational effort to solve the problem in affordable times with regular computers.

7.3 Computational Analysis

In this section we analyze the results of our tests over the 12 hard instances (from $n = 30$ to 100 variables) composing the suite 1 of the benchmark proposed in [32]. These instances belong to the SAT phase transition of difficulty, where hardest instances are located, since they verify that $m = 4.3 * n$ [182] (being m the number of clauses).

Next sections discuss the results in a structured way. All algorithms have been tested in terms of efficiency (average number of evaluations to reach a solution –AES–) and efficacy (success rate –SR–). The details are shown in Tables 7.2, 7.3, 7.5, and 7.6; in Fig. 7.1 we plot a summary on the success rate results. All the results are average values after 50 independent runs of every algorithm on every problem instance. Statistical tests have been used to ensure the validity of our claims. For this, a confidence level of 95% is represented in the tables with a '+' symbol ('•' means no statistical differences).

7.3.1 Effects of Combining a Structured Population and an Adaptive Fitness Function (SAW)

In this section we justify the election of both the structured (cellular) population and the use of the stepwise adaptation of weights function (SAW) in our cMAs. Our goal is to show that they two are the minimum and common requirements to get solutions with a quality similar as that reported in past contributing works. With that purpose, we study the behavior of JCell.DPX_BM

Table 7.2. Effects of structuring the population combined with SAW in the fitness. Average number evaluations to reach a solution (AES)

Instance size (n)	#	genGA	JCell.DPX_BM_cl	JCell.DPX_BM	Test
30	1	49801.0	**3415.7**	7438.0	+
30	2	1135843.2	—	**502208.4**	+
30	3	440886.2	**3024.0**	80029.4	+
40	4	855286.3	31932.0	**13829.8**	+
40	5	66193.9	**4861.4**	9391.7	+
40	6	1603008.0	—	**519868.8**	+
50	7	473839.4	14356.8	**13081.0**	+
50	8	1076077.4	**84088.8**	95379.8	+
50	9	1333872.0	—	**524164.1**	+
100	10	—	—	**601488.0**	+
100	11	—	223310.8	**165484.8**	+
100	12	—	**245232.0**	392871.8	+

(a cGA using SAW, and implementing generic recombination and mutation operators –DPX and BM, respectively–), compared to its generational version –non-structured population– (genGA), and JCell.DPX_BM using a classic fitness function for SAT (JCell.DPX_BM_cl), i.e., counting the number of the clauses satisfied by the potential solution.

Our results are given in Table 7.2, wherein we show for every instance: its size, its identifier (#), and the average number of evaluations needed to find the solution (AES) for the three algorithms (best values are **bolded**). Additionally, the number of runs in which the solution was found (success rate) is displayed in Fig. 7.1. After applying statistical tests to the results in Table 7.2, we obtained that there are statistically significant differences in all the instances.

One can see that genGA is the algorithm reporting the worst AES results, and JCell.DPX_BM is worse than JCell.DPX_BM_cl with statistical significance only in 2 instances (numbers 1 and 5). In terms of SR (see Fig. 7.1), JCell.DPX_BM is better than the other two algorithms (higher efficacy) for the 12 instances. Moreover, only JCell.DPX_BM is able to find the optimal solution for every problem. Hence, from these results we can conclude that JCell.DPX_BM, the algorithm with structured population and using SAW, is the best of the three compared ones, both in terms of efficacy (SR) and efficiency (AES).

From Table 7.2 and Fig. 7.1, it stands out that the tested cGA markedly improved its results when using the SAW fitness function. Because of that, all the algorithms studied in the following sections have been implemented using SAW. Although the authors are aware that the use of SAW does not necessarily improve the performance on other algorithms, the SAW fitness function has been implemented in GRAD, SA, and WSAT as also suggested in [15]. It is made in order to make fair comparisons between them and our cMAs.

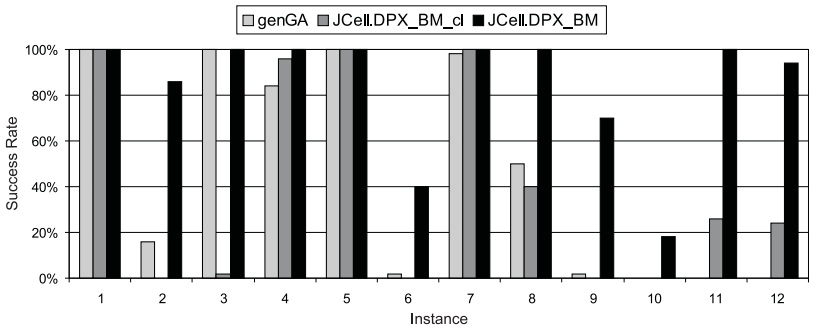


Fig. 7.1. Effects of structuring the population and using SAW. Success rate (SR)

7.3.2 Results: Non Memetic Procedures for SAT

In this section we study the behavior of GRAD, SA, and WSAT. Additionally, we test the behavior of two different cGAs without any local search: the previously studied JCell.DPX_BM, and also JCell.UCR_UCM, which is the result of hybridizing the former with recombination and mutation operators specifically designed for SAT (UCR and UCM). The parameters used are those of Table 7.4, but in this case $P_{\text{LS}} = 0.0$ (i.e., there is no LS). In Table 7.3 we show our results. The first issue we want to emphasize is that only WSAT is able to solve the problem in every run for all the instances. As an interesting detail, in [120] WSAT only solved the problem in a 80% of the runs in the case of the largest instances (with $n = 100$), so we think to have a better implementation of WSAT here.

When comparing the three basic LS, it can be seen in Table 7.3 that SA obtains the worst results, both in terms of efficacy and efficiency (with statistical confidence, except for instance number 10). GRAD is similar to WSAT in efficacy, but it needs a larger number of fitness function evaluations to find the solution (lower efficiency), except for instance 6 (significant values obtained in instances 3, 4, 7, 8, and 10 to 12). Hence, we can conclude that WSAT is the best of the three heuristics for the studied test-suite, followed by GRAD. We also conclude on the superiority of problem dependent algorithms versus generic patterns of search like SA, at least at a moderate level of customization effort.

With respect to the two cGAs, Table 7.3 states the opposite result: the cGA with generic operators outperforms the one including tailored mutation and recombination. This is clear since JCell.UCR_UCM reports a larger AES than JCell.DPX_BM (statistical confidence for all the instances, except 2). Moreover, JCell.UCR_UCM also performs a lower hit rate (SR) than JCell.DPX_BM in general. In addition, JCell.UCR_UCM is not able to find the optimum in any of the 50 runs made for 6 out of the 12 instances. Probably, the reason for the mentioned poor behavior of JCell.UCR_UCM with respect to JCell.DPX_BM is that both UCR and UCM perform a too intensive exploitation of the search space, resulting in an important and too fast loss of diversity in the population; this makes the algorithm get stuck in local optima frequently.

Table 7.3. Performance of the basic algorithms

#	GRAD		SA		WSAT			JCell.DPX_BM		JCell.UCR_UCM		
	SR	AES	SR	AES	SR	AES		SR	AES	SR	AES	
1	1.00	203.56 ±219.73	1.00	685.22 ±844.74	1.00	**143.64** ±120.32	+	1.00	7438.04 ±3234.60	1.00	104100.48 ±121339.96	+
2	1.00	9681.06 ±9483.55	1.00	63346.60 ±93625.68	1.00	**8575.66** ±9244.08	+	0.86	502208.37 ±491034.68	0.10	697564.80 ±663741.62	•
3	1.00	8520.38 ±7724.11	1.00	16833.44 ±11002.84	1.00	**3984.34** ±4112.95	+	1.00	80029.44 ±54664.78	0.98	269282.94 ±223859.24	+
4	1.00	619.94 ±584.88	1.00	2173.62 ±2076.57	1.00	**199.56** ±193.56	+	1.00	13829.76 ±7801.37	0.10	1364688.00 ±500365.96	+
5	1.00	324.46 ±332.19	1.00	1202.86 ±1045.82	1.00	**103.66** ±88.02	+	1.00	9391.68 ±2478.37	1.00	249137.28 ±236218.19	+
6	1.00	**14368.98** ±13954.02	0.86	271701.47 ±418129.55	1.00	14621.04 ±18617.88	+	0.40	519868.80 ±552312.72	0.00	—	—
7	1.00	496.58 ±359.60	1.00	1614.76 ±1252.34	1.00	**200.84** ±154.81	+	1.00	13080.96 ±3346.94	0.10	1005494.40 ±721439.61	+
8	1.00	1761.74 ±1989.06	1.00	9512.84 ±10226.14	1.00	**793.38** ±870.94	+	1.00	95379.84 ±125768.68	0.00	—	—
9	1.00	82004.84 ±63217.93	1.00	201612.46 ±266218.97	1.00	**77696.42** ±75769.23	+	0.70	524164.11 ±432005.51	0.00	—	—
10	0.94	726522.51 ±525423.23	0.84	510006.12 ±419781.41	1.00	**189785.14** ±198738.78	+	0.18	601488.00 ±364655.49	0.00	—	—
11	1.00	5508.26 ±5940.96	1.00	18123.00 ±20635.35	1.00	**1501.74** ±1264.80	+	1.00	165484.80 ±190927.59	0.00	—	—
12	1.00	8920.38 ±9111.02	1.00	25539.84 ±22393.45	1.00	**1388.92** ±1308.27	+	0.94	392871.83 ±443791.69	0.00	—	—

As a final conclusion, we can claim from Table 7.3 that the results of the two cGAs without explicit LS are always worse than those of the LS heuristics, both in terms of efficiency and efficacy. The only best results of the table are those obtained by WSAT. Since we suspect that these results are too linked to the instances (specially to their "small" size) we will enlarge the test set at the end of next subsection with harder instances in order to get conclusions on the problem class if possible (and detect the influence in our results of any bias on the instances).

7.3.3 Results: Cellular Memetic Algorithms

In this section we study the behavior of a large number of cMAs with different parameterizations. As it can be seen in Table 7.4 (wherein details of the cMAs are given) we here hybridize the two simple cGAs of the previous section with three distinct local search methods (GRAD, SA, and WSAT). These local search methods have been applied in two different ways: (i) executing an intense local search step ($10 \times n$ fitness function evaluations) to a given percentage of the individuals (called *intensive*), or (ii) applying a light local search step to all the individuals, consisting in making only 20 fitness function evaluations in these methods (called *light*).

Results are shown in Tables 7.5 and 7.6. Comparing them with those of the cGAs of Table 7.3, we can see that the behavior of the algorithm is generally improved both in efficiency and efficacy, specially when using GRAD and WSAT. Hence, it is clear that the three local search methods used (generic and specific) help the algorithm for getting out from local optima.

Table 7.4. General parameterization for the studied cMAs

	JCell.DPX_BM+ GRAD,SA,WSAT	JCell.DPX_BM_i+ GRAD,SA,WSAT	JCell.UCR_UCM+ GRAD,SA,WSAT	JCell.UCR_UCM_i+ GRAD,SA,WSAT
Local Search	Light $P_{LS} = 1.0$	Intensive $P_{LS} = 1.0$/popsize	Light $P_{LS} = 1.0$	Intensive $P_{LS} = 1.0$/popsize
Mutation	Bit-flip ($P_{bf} = 1/n$), $P_m = 1.0$		UCM, $P_m = 1.0$	
Crossover	DPX, $P_c = 1.0$		UCR, P_c	
Pop. size	144 individuals			
Selection	Itself + binary tournament			
Replacement	Replace_if_Better			
Stop cond.	Find a solution or reach 100,000 generations			

Table 7.5. Results for the proposed hybridizations of JCell.DPX_BM

	JCell.DPX_BM						JCell.DPX_BM_i					
#	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
1	1.00	3054.2 ±392.2	1.00	29474.5 ±583.4	1.00	2966.1 ±19.2	1.00	1072.8 ±1112.6	1.00	9649.6 ±25809.9	1.00	**569.9** ±302.8
2	1.00	33598.7 ±51766.6	1.00	195397.6 ±295646.3	1.00	32730.4 ±49353.2	1.00	50886.2 ±44167.7	0.90	559464.3 ±437996.2	1.00	**30885.5** ±22768.8
3	1.00	14761.2 ±24935.1	1.00	33005.4 ±6306.3	1.00	**4104.5** ±3325.1	1.00	20385.8 ±20115.7	1.00	255902.5 ±275734.7	1.00	9418.4 ±10239.6
4	1.00	5018.6 ±2397.8	1.00	31618.8 ±152.9	1.00	3972.9 ±1343.8	1.00	2573.4 ±2497.7	1.00	49310.9 ±64714.9	1.00	**794.7** ±693.7
5	1.00	3575.6 ±1131.5	1.00	31052.9 ±282.7	1.00	3008.3 ±8.4	1.00	1586.0 ±1757.9	1.00	13354.0 ±36668.5	1.00	**628.6** ±437.9
6	0.96	181863.6 ±343020.8	0.96	434235.9 ±519011.4	1.00	81966.1 ±114950.0	1.00	94046.4 ±114105.9	0.72	654160.4 ±476411.6	1.00	**41619.4** ±47466.8
7	1.00	5945.8 ±2416.8	1.00	33621.6 ±7313.2	1.00	4822.6 ±1364.9	1.00	2342.6 ±2972.9	1.00	37446.4 ±70165.5	1.00	**850.8** ±527.5
8	1.00	14930.8 ±7644.5	1.00	47688.6 ±15925.1	1.00	7138.3 ±3957.5	1.00	5164.5 ±5786.7	1.00	195816.2 ±155018.9	1.00	**2097.6** ±1886.8
9	0.80	787149.2 ±528237.4	0.50	720491.5 ±597642.6	1.00	600993.9 ±443475.3	0.82	963177.2 ±585320.7	0.34	883967.7 ±633307.9	1.00	**187814.5** ±148264.1
10	0.06	797880.3 ±824831.9	0.04	1209394.0 ±90058.5	0.06	1189559.7 ±374193.7	0.04	1302489.0 ±346149.9	0.10	1363627.4 ±368403.3	0.80	**792051.2** ±491548.4
11	1.00	58591.3 ±18897.3	1.00	1039910.2 ±205127.9	1.00	35571.0 ±9243.6	1.00	12539.8 ±10851.1	1.00	357207.9 ±422288.9	1.00	**2466.3** ±1846.4
12	0.96	70324.9 ±32808.8	0.98	1051351.2 ±174510.4	1.00	45950.2 ±19870.7	1.00	20018.2 ±19674.3	0.98	409492.6 ±425872.3	1.00	**3196.9** ±2938.3

If we compare the studied cMAs in terms of the way the local search method is applied (intensive or light), we conclude that the intensive case always obtains better results than the other when hybridizing the algorithm with one specific heuristic (either GRAD or WSAT). On the contrary, this is not always true when using SA, since the intensive version of SA only outperforms its lighter counterpart in 9 out of the 24 tests. Hence, cGAs hybridized with specialized LS have a better performance than the one using SA (generic) on average, and specially when intensive search is used. All these comparisons are statistically significant in 58 out of the 65 cases (for them all the cMAs obtained the solution in almost 100% of the runs).

As an interesting exception, we want to remark the nice efficacy of JCell.UCR_UCM+SA for instances 11 and 12 with respect to JCell.UCR_UCM hybridized with GRAD and WSAT, since these two last cMAs are not able to find the optimal solution in any run. The reason is probably a too high intensification of GRAD and WSAT performed on the population (remind that they are still merged with UCR and UCM), guiding the algorithm towards a local optimum quickly.

Table 7.6. Results for the proposed hybridizations of JCell.UCR_UCM

#	JCell.UCR_UCM						JCell.UCR_UCM_i					
	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
1	1.00	2981.2 ±18.9	1.00	29253.2 ±474.8	1.00	2953.1 ±27.8	1.00	1239.1 ±1467.1	1.00	21710.7 ±46248.4	1.00	**748.8** ±404.1
2	1.00	20294.7 ±23868.0	1.00	187088.9 ±281719.9	1.00	**14879.6** ±18766.3	1.00	58842.3 ±62944.9	1.00	686104.1 ±527621.8	1.00	31457.6 ±33033.8
3	1.00	4048.0 ±2832.1	1.00	41269.5 ±58635.5	1.00	**3641.2** ±1861.2	1.00	25086.8 ±24428.4	1.00	280148.0 ±217802.8	1.00	13614.9 ±13134.6
4	1.00	7853.8 ±9207.1	1.00	31527.8 ±187.2	1.00	3472.5 ±1773.7	1.00	2299.6 ±2937.4	1.00	63190.8 ±110063.6	1.00	**779.4** ±408.9
5	1.00	3466.3 ±1781.8	1.00	30893.9 ±246.8	1.00	2976.1 ±19.9	1.00	1193.1 ±1198.5	1.00	18722.7 ±56165.8	1.00	**624.7** ±369.2
6	1.00	379489.9 ±351593.1	1.00	274977.7 ±389332.4	1.00	162737.1 ±180706.5	1.00	86780.6 ±71185.9	0.94	849405.5 ±584901.9	1.00	**57997.9** ±48455.4
7	1.00	7335.1 ±7980.3	1.00	31715.6 ±134.0	1.00	3532.0 ±1807.9	1.00	1639.8 ±2297.5	1.00	96672.3 ±158359.2	1.00	**678.2** ±507.7
8	1.00	82967.7 ±76765.2	1.00	46418.0 ±15867.9	1.00	27090.5 ±30079.4	1.00	6747.4 ±8070.6	1.00	291700.2 ±225526.0	1.00	**1694.4** ±1619.9
9	0.42	1089600.1 ±642627.4	0.64	1365366.8 ±559506.2	0.56	694014.5 ±548185.0	0.92	566331.3 ±476381.3	0.48	1155717.8 ±529793.2	1.00	**305306.2** ±323215.9
10	0.00	—	0.00	—	0.00	—	0.76	885961.2 ±630092.4	0.16	1099241.9 ±768918.5	1.00	**425377.6** ±415069.5
11	0.00	—	0.64	1743364.4 ±190880.9	0.00	—	1.00	10560.4 ±11327.4	0.90	695508.1 ±855309.5	1.00	**2980.8** ±3334.6
12	0.00	—	0.40	1778928.5 ±200497.9	0.00	—	1.00	16623.6 ±18137.9	0.94	504324.6 ±770231.7	1.00	**3949.3** ±4646.0

Let us now compare the best algorithm in Table 7.3 (WSAT) versus the best one included in Tables 7.5 and 7.6 (JCell.UCR_UCM_i+WSAT). These two algorithms are the best out of all the studied ones in terms of efficiency and efficacy. The two algorithms find the optimal solution in 100% of the runs (SR=1.0 for every instance), but JCell.UCR_UCM_i+WSAT obtains worse (numerically higher) results than WSAT in terms of AES (with statistically significant differences).

Although we expected a hard comparison against the best algorithm in literature (WSAT), this was not the case, since we got similar accuracy and only slightly worse efficiency in our tests. Since we suspected this holds only in the smaller instances we decided to test these two algorithms with larger instances in order to check if the cMA is able to outperform the state of the art WSAT in harder problems. For that, we have selected the 50 instances of 150 variables from the suite 2 of the same benchmark [32] studied before. The results with the larger instances show that JCell.UCR_UCM_i+WSAT solved the problem at least once (of 50 executions) in 26 out of the 50 instances composing the benchmark, while WSAT found the solution for the same 26 instances and 4 more ones (the optimum was found only once in these 4 instances). Hence, WSAT is able to find the optimum in a larger number of instances, with an average hit rate of 38.24%, which is quite close to 36.52%, the value obtained by JCell.UCR_UCM_i+WSAT.

Table 7.7. Efficacy (SR) of algorithms when evaluated on SAT

Algorithm	n = 30	n = 40	n = 50	n = 100
SAWEA	1.00	0.93	0.85	0.72
RFEA2	1.00	1.00	1.00	0.99
RFEA2+	1.00	1.00	1.00	0.97
FlipGA	1.00	1.00	1.00	0.87
ASAP	1.00	1.00	1.00	1.00
WSAT [120]	1.00	1.00	1.00	0.80
GRAD	1.00	1.00	1.00	0.98
WSAT	1.00	1.00	1.00	1.00
JCell.DPX_BM_i+WSAT	1.00	1.00	1.00	0.93
JCell.UCR_UCM_i+WSAT	1.00	1.00	1.00	1.00

Moreover, the average solution found for this benchmark (the optimal solution is 645 for all the instances) is 644.20 for JCell.UCR_UCM_i+WSAT and 643.00 for WSAT, so the cMA is more accurate than WSAT for this set of instances. Finally, if we compute the average AES for the instances solved (at least once) by the two algorithms we can see that the cMA (AES = 364,383.67) is in this case more efficient than WSAT (AES = 372,162.36). Hence, as we suspected, the cMA outperforms WSAT for this set of larger and more difficult problems. In fact, all these results represent the new state of the art since "our" WSAT is better than the one previously reported in the literature.

7.3.4 Comparison Versus Other Algorithms in the Literature

In this section we compare some of our results with those of the algorithms studied in [120], which were tested with the same benchmark we used in this chapter. The comparison is shown in Table 7.7, where only the efficacy of the algorithms is shown, because the efficiency is measured in [120] as the number of bit flips to solution (AFS) instead of the number of function evaluations (AES).

As it can be seen in Table 7.7, only ASAP [120], our implementation of WSAT, and JCell.UCR_UCM_i+WSAT are able to find the solution in every run for all the instances. Anyway, all the results are really good, since most of the algorithms of the table have a very high efficacy for the tested benchmark, usually with hit rates over 90%.

The difference in behavior of our WSAT and that studied in [120] are both the noise probability and the termination condition. On the one hand, we use a noise probability of 0.5, while Gottlieb et al. do not specify in [120] the value they use. On the other hand, our algorithm stops when 2 million function evaluations are made (at most), while the termination condition in the algorithm of Gottlieb et al. is to reach a maximum of 300,000 flips. Although the number of evaluations made by our algorithm, on average, is higher than

the number of flips in our implementation of WSAT (since we use SAW, the best stored individual has to be re-evaluated) we cannot clearly compare them under this metric. However, in an effort to give some hints on their differences we run additional experiments to obtain the SR values when setting the termination condition to 300,000 evaluations (less than 300,000 flips should be made). The numerical values for SR are 1.0, 1.0, 0.99, and 0.95 for the groups of instances with n = 30, 40, 50, and 100, respectively. These values are still higher (larger efficacy) than those obtained by Gottlieb el al., which is another clear indication that our proposal is comparable to theirs.

7.4 Conclusions

In this chapter we have proposed several ways of creating cMAs, and have analyzed the behavior of 3 LS methods, 2 basic cGAs, and 12 cMAs on the 3-SAT problem. These cMAs are the result of hybridizing the two cGAs with the 3 LS, applied with different parameterizations reinforcing diversification or intensification. Two LS techniques, WSAT and GRAD (this one specially developed in this work), are specifically designed for SAT, while SA is a generic representative of trajectory based search method used in its basic form.

We have seen that the results of the proposed basic cGAs (without local search) are far from those obtained by the three studied LS methods by themselves, and this is a clear indication of the need of exploitation procedures for the problem at hands, i.e., a perfect context to create cMAs. After hybridizing these basic cGAs with a local search operator in their loop, the resulting cMAs substantially improved the behavior versus the original cGAs. Thus, the hybridization step helps the cMAs to avoid the local optima in which the simple cGAs get stuck. For smaller instances, the best of the tested cMAs (JCell.UCR_UCM_i+WSAT) is as accurate as the best reported algorithm (WSAT) but slightly less efficient.

After these results, we studied the behavior of WSAT and JCell.UCR_UCM_i+WSAT (the two best resulting algorithms) with a harder set of larger instances. The results confirm our feelings after the preliminary results on the smaller instances: the cMA is more accurate and efficient than WSAT for these harder instances. After such a deep set of studies, our final claims contrast with those of Gottlieb et al., who concluded in [120] that "A preliminary experimental investigation of EAs for constraint satisfaction problems using both an adaptive fitness function (based on f_{SAW}) and local search indicates that this combination is not beneficial". We found that this claim does not hold at least when using structured algorithms for solving large instances.

8

Design of Parallel Cellular Genetic Algorithms

I was married at 16, a father at 17, and divorced at 18.

Plácido Domingo (1941 -) – Spanish Opera Singer

Most researchers want their results to be accurate, and specially, they want these results now, immediately, or as soon as possible. In this context, parallel algorithms come to scene, allowing the utilization of several CPUs for solving a given problem.

When one addresses the field of parallel EAs a big number of implementations and techniques spring out [6, 43, 196, 241]. The reason for such a success is firstly due to the inherent parallelism inside EAs, since most of the variation operators can be independently run in parallel. An important additional fact is that parallel EAs often leads not only to a faster algorithm, but to an algorithm performing an outstanding larger numerical performance compared to sequential equivalent approaches [27, 114]. The basic reason for such results is that of using a spatially structured population, either in the form of islands [243] (distributed EAs) or in the form of grid (mesh) [176] (cellular EAs). As a consequence of such success many authors have reported the higher performance of parallel algorithms versus their sequential counterparts from the very beginning of the research in evolutionary algorithms [114].

At their start, cellular EAs (cEAs) appeared as a new algorithmic model intended to profit from the hardware features of massively parallel computers, specially those using several hundreds/thousands processors. On this hardware, a basic cEA should assign a single individual (tentative solution) to each processor in an independent fashion. Since a minimum overhead is very important in such systems, simple interactions with nearby processors that the cEA employs lead to is a perfect numerical model for this kind of computers. With time, this sort of massively parallel computers became less and less popular, and at this moment what remains is only the numerical model of search by overlapped neighborhoods, regardless of what type of machine is used for their implementation.

Nowadays, the reader can find a set of different parallel cEA implementations, either in multiprocessor computers, clusters of distributed memory, and even concurrent implementations on sequential computers (see Chap. 2). Although they are different implementations, the underlying numerical model

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_8, © Springer Science+Business Media, LLC 2008

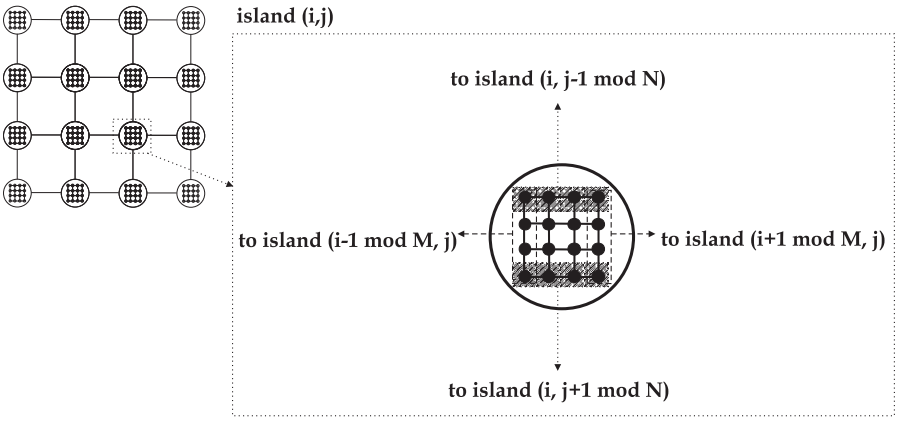


Fig. 8.1. Parallel distributed schema of work for the meta-cellular GA

is the same: that of a cellular EA. In this chapter we are presenting two new parallel models in which concepts are merged from the two main fields: coarse and fine grain procedures.

In Sect. 8.1 we describe a new model called meta-cellular algorithm, in which the population is divided into a set of islands, all of them interconnected in a toroidal grid, every island performing a cellular GA. In addition, in Sect. 8.2 we discuss an adaptation of an early technique [5] called dcGA; this search procedure uses also distributed islands interconnected in a ring, again every island running a separate cGA. This last search technique will be illustrated by solving very large instances of VRP on a *grid* platform made of 125 computers. Finally, in Sect. 8.3 we sum up the main conclusions of this chapter.

8.1 The Meta-cellular Genetic Algorithm

The goal of this section is that of presenting a parallel cGA targeted to distributed systems, that we call *meta-cellular genetic algorithm* or meta-cGA. In this technique, the population is divided among all the available processors but keeping unchanged the numerical behavior of a regular sequential cGA. At the beginning of every iteration all processors are sending the individuals located at the north, south, east, and west of their grid to their neighboring islands situated, in turn, to the north, south, east, and west, respectively (as shown in Fig. 8.1). The inter-island cellular algorithms are canonical. To illustrate its behavior we here include the results of a sequential cGA and various other parallel models.

Table 8.1. Average results for all the parallel models

Alg.	success %	# evals	time	Alg.	success %	# evals	time
Sec.	60%	97671	19.12				
MS2	61%	95832	16.63	idGA2	41%	92133	9.46
MS4	60%	101821	14.17	idGA4	20%	89730	5.17
MS8	58%	99124	13.64	idGA8	7%	91264	2.49
MS16	62%	96875	12.15	idGA16	0%	—	—
dGA2	72%	86133	9.87	meta-cGA2	**85%**	92286	10.40
dGA4	73%	88200	5.22	meta-cGA4	83%	94187	5.79
dGA8	72%	**85993**	2.58	meta-cGA8	83%	92488	2.94
dGA16	68%	93180	**1.30**	meta-cGA16	84%	91280	1.64

8.1.1 Parameterization

We here describe the parameters used for the algorithms mentioned in the previous section. Our aim is not to get competitive final results, but instead to compare the algorithms to offer the reader with a feeling of the basic power of each type of search; this of course means that those parameters can be fine tuned for higher performance. The global population size is 800 individuals; in the parallel methods each processor is having a part of this population, i.e., $800/n$ individuals, being n the number of processors. All the algorithms use a single point recombination operator (–SPX–) with an application probability of 0.7, as well as a simple bit-flip mutation applied with probability 0.2.

In the distributed algorithms the migration step is undertaken every 20 iterations in an asynchronous manner. The topology of migration is a unidirectional ring of islands, and the exchanged information is one single individual locally selected in every island in a uniformly random fashion. Every island integrates the incoming migrant individual only if it is better in fitness than its worst present solution.

All the experiments have been run on a cluster of 16 PCs Pentium 4 2.8 GHz with Linux linked by a Fast Ethernet network. Since the algorithms are inherently nondeterministic we perform 100 independent runs in order to get meaningful conclusions from a statistical point of view.

8.1.2 Analysis of Results

Let us start with a comparison of the basic algorithm against other parallel models of GA on the well known problem SAT (whose details can be accessed in Appendix A) initially proposed in [140]. All the SAT instances are made of 100 variables and 430 clauses, thus belonging to the interesting phase transition of this problem (see again Appendix A).

Our discussion starts by analyzing the contents of Table 8.1, where column *success %* contains the rate of executions finding the optimum, column *# evals* reports the number of required evaluations and column *time* indicates the execution time (in seconds) for all the algorithms: the sequential GA (Sec.), a

Table 8.2. Weak *speedup* for the considered algorithms

Alg.	Speedup			
	n = 2	**n = 4**	**n = 8**	**n = 16**
MSn	1.14	1.34	1.40	1.57
idGAn	**2.02**	**3.69**	**7.67**	—
dGAn	1.93	3.66	7.41	**14.70**
meta-cGAn	1.83	3.30	6.50	11.65

master-slave GA (MS), a distributed GA in which the islands are not communicating at all among them (idGA), a distributed method in which the island GAs do collaborate among them (dGA), and finally the new proposed model meta-cGA. All parallel algorithms have been evaluated on 2, 4, 8, and 16 processors. We also report in Table 8.2 the resulting *speedup*. We use the *weak* definition of *speedup* [25], i.e., we compare the parallel implementation runtime with respect to the sequential one. Best values are marked in **boldface** in the table.

The interpretation of the results in Table 8.1 are indicative of several interesting facts. Firstly, let us analyze every model separately. As could be expected, the numerical behavior of the master-slave and the sequential models is the same, both in visited points and percentage of success. Of course, the master-slave model runs in parallel and this is the reason for requiring a lower runtime, although we can notice that this model is not very good in time gains as the number of processors grows (Table 8.2). In this case, not only the model, but the short evaluation time of the problem is small in comparison to the considerable communication overhead.

The idGA search model throws also common sense results: the time is reduced and the *speedup* is good, almost linear (Table 8.2). On the contrary, since the search is not collaborative, the percentage of executions finding the optimum is severely penalized (even zero with 16 processors). This model is equivalent to running small algorithms, which is expected to be good in efficiency (no communications) but bad in efficacy (progressively smaller populations with the growing number of processors with no collaboration).

In general, we can observe that the distributed GAs are better than the sequential ones, both numerically and in time, since they get a higher rate of success with a lower number of overall evaluations and in a shorter time. The *speedup* is in general good (quasi-linear) with a perceptible degradation as the number of processors increases.

Finally, if we now consider the evaluated cGAs, we can conclude that they are the globally best in these experiments, since they are very good in showing a high percentage of success, a main goal in solving complex problems. In fact, they report also very short execution times, just slightly worse than those of the distributed GAs.

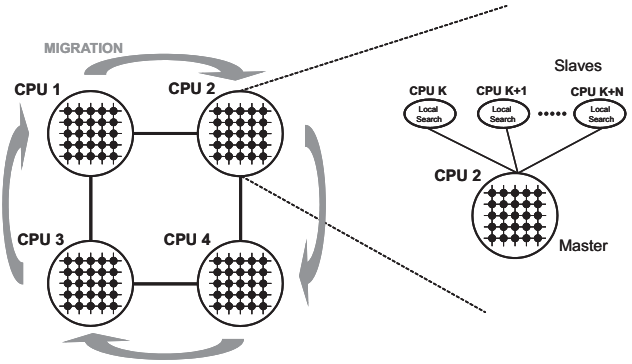


Fig. 8.2. Components of PEGA

8.2 The Distributed Cellular Genetic Algorithm

In this section we deal with a new parallel cGA we have proposed named PEGA (Parallel cEllular Genetic Algorithm) [75]. For the design of PEGA we adapted the two main strategies for parallelization: coarse and fine grain approaches. Therefore, PEGA exhibits a parallel technique in which the population is structured in two levels and in two different forms. As it can be seen in Fig. 8.2, in the first level the population is structured in several islands (coarse grain) connected in a ring (each island thus having a single neighbor). In the second level, in order to enhance the search abilities of every island, we run a (fine grain) cGA inside each island. The basic ideas and motivation for such a model were initially proposed in [5]. The contribution of PEGA with respect to these initial models resides in that the separated cGAs act each one as a master process that delegates costly tasks as evaluations to other slave processors.

PEGA offers as a consequence the potential to be executed in grid platforms with hundreds of computers, not only because of its design, but also because it has been implemented with ProActive [2]. ProActive is a grid library written in Java targeted to write programs for the grid. ProActive endows its users with simple mechanisms for the general management of remote collaborating objects (called *active objects*) running in separate processors. Access to active objects looks like access to local objects, and ProActive is in charge of the underlying tasks for effectively communicating information and performing activities by internally using Java RMI.

In short, our contribution here has been to provide the research community with a hybrid cellular GA suited for grid environments. It is called hybrid because it incorporates knowledge of the problem in all its variation operators. This algorithm has been used to solve the largest known instances of VRP existing in the literature [162], with very interesting results, since PEGA has improved the best solution known for most of them.

In Sect. 8.2.1 we now discuss the used parameters, as well as the mentioned instances of VRP used for testing PEGA. In Sect. 8.2.2 we later present and analyze the results of our study.

8.2.1 Parameterization

In recent years, Li, Golden, and Wasil proposed in [162] a new set of CVRP instances whose main interest is the very high number of clients compared to other existing benchmarks in Internet. This set of instances is known as VLSVRP (*Very Large Scale VRP*), all of them are restricting the maximum length of the resulting routes. The size of such VLSVRP benchmark ranges from 560 up to 1200 clients, while the rest of benchmarks used until now range from 50 up to 199 clients (CMT [47]), or from 200 up to 483 clients (like in [109]). We should warn that the size is not the only source for difficulties in VRP, but definitely it has an important influence in the resulting difficulty when added to the rest of restrictions found in most benchmarks and of course provoke larger execution times in the algorithms used to solve them. VLSVRP has been created by using a program generator [162], and all of them share a nice geometrical shape when plotted, which can help in understanding the results got by the used algorithms.

As previously mentioned, PEGA is running a set of islands, each one executing a cGA. To solve VRP these cGAs are canonical but enhanced with a local search method similar to that used in Chap. 13. This local search is applied to all the individuals in the cGA, resulting in the operation of higher computational requirements, and thus the obvious candidate to be executed in the slave processors. Before presenting the parameterization let us discuss some more details of the algorithm to clearly make this work easy to reproduce and to extend in the future.

- **Representation of the solutions.** The used representation for the individuals of the algorithm is the well known GVR (*Genetic Vehicle Representation*) [206]. In GVR the genotype is a permutation of integers (client IDs) merged with some information to point out where every route is finishing inside the permutation. Our algorithm always manipulates feasible individuals, and thus if the maximum capacity of a vehicle is violated or a route length exceeds the maximum allowed value, the algorithm proceeds to split the route in two or more new routes, all of them feasible.
- **Generation of the initial population.** Initial individuals are generated in a random way, and immediately repaired to eliminate any unfeasible route appearing in them (according to the mechanism presented in [206]). Therefore, all the initial solutions are valid and no restriction is violated by them.
- **Recombination operator.** The recombination used in PEGA is the *generic crossover*, originally proposed in [206]. This operator is a capital source for diversity during the search, a main issue in solving VRP as our

Algorithm 8.1 Pseudo-code of the generic recombination used in PEGA

1. // Let I_1 and I_2 the parents selected from the neighborhood;
2. *Select* one random sub-route $SR = \{a_1, \ldots, a_n\}$ from I_2
3. *Search* for the client $c \notin SR$ located geographically closer to a_1
4. *Delete* all clients in I_1 who also are in SR
5. The offspring is computed after inserting SR in the genotype of I_1 such that a_1 be located immediately after c

own experience clearly advise [11, 13, 14], since premature convergence to local optima is a big difficulty in this application. This operator works in a very special manner, since it not only performs recombination of the material contained in the parent solutions, but also introduces new information in the offspring (see Fig. 8.3). In Alg. 8.1 we show a pseudo-code of this operator.

- **Mutation.** The used mutation is made of four basic mutation components, one of them being selected to be applied according to its own probability on every mutation application (see [206]). The effect of such four operations is to modify one route, or to exchange clients between different routes, as well as to add or to delete routes. See a description of the component operations in Fig. 8.4:
  - **Swap.** It consists in exchanging to clients, either intra or inter routes. The two clients are selected randomly.
  - **Inversion.** This operator works inside a single route; once two bounding clients have been selected, all the sub-permutation of clients between they two is inverted.
  - **Insertion.** This operation randomly selects one client and a new random position where to insert it, either in the same route or in a different one.
  - **Dispersion.** Similar to insertion, but using a sub-route instead of just one client.
- **Local Search.** The local search used in PEGA applies 50 steps of a basic *1-Interchange* [200]; after this, it applies 50 steps of *2-Opt* [55] on every

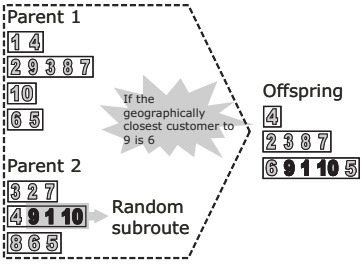


Fig. 8.3. The used recombination: *generic crossover*

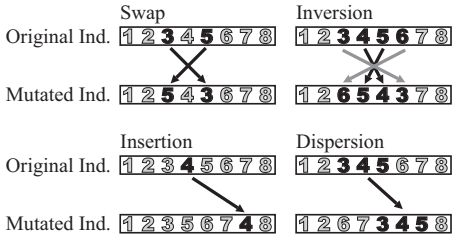


Fig. 8.4. How mutation is applied

Table 8.3. Parameterization for the experiments with PEGA

Population size	Islands: 100 individuals (10×10)
	Total: 400 individuals (4 islands)
Neighborhood	NEWS
Parent selection	Current individual + binary tournament
Recombination	*Generic crossover*, $p_c = 1.0$
Mutation of	Swap ($p_{\text{int}} = 0.05$), Inversion ($p_{\text{inv}} = 0.1$),
individuals	Insertion ($p_{\text{ins}} = 0.05$), and Dispersion ($p_{\text{disp}} = 0.15$)
Replacement	Replace_if_Better
Local search	1-Interchange + 2-Opt,
	50 steps per method
Freq. of migration	Every 10^4 evaluations
Stop Condition	Maximum of $5 \cdot 10^5$ evaluations per island

route of the previously worked solution. The *1-Interchange* works by exchanging two clients between two routes, or either by inserting one client in a different route of the solution. On its own side, the 2-Opt operator always works inside a single given route, by removing two edges from the route and reconnecting the bounding clients of the resulting sub-route in the other possible way. Since these two methods are deterministic they stop when no improvement is achieved, thus allowing a considerable reduction in the execution time.

PEGA has been initially engineered to solve VLSVRP (although nothing prevents its utilization for other problems). Since the target problems are complex and showing a high dimension we targeted PEGA for a grid of computers; in our experiments we use a maximum of 125 computers of different features (e.g., either PCs or SUN workstations). PEGA has been codified in Java using ProActive to deal with the grid.

The used parametrization is described in Table 8.3. In particular, it runs 4 islands in a ring, each one exchanging one single individual with its neighboring island every 10^4 evaluations. In every island a canonical cGA is working on a mesh composed of 10×10 individuals, with an additional local search method similar to that used in Chap. 13. In such cGA, one of the parents is selected according to a binary tournament in the neighborhood of the presently considered individual (NEWS), while the other parent is the mentioned present individual. Recombination is always applied according to the *generic crossover* method, and the resulting individual is mutated by applying on it the operators swap, inversion, insertion, and dispersion at probabilities 0.05, 0.1, 0.05, and 0.15, respectively.

Afterwards, the offspring is in addition improved by a local search using 50 steps of *1-Interchange* [200] (inter-route client exploration) followed by another 50 steps of *2-Opt* [55] (intra-route exploitation). The offspring resulting from this local search is replacing the present individual only if its fitness is better than that of the present individual.

Table 8.4. Results of applying PEGA to VLSVRP

Instance	Size	MSC	Best	Average	Time (h)
VLS21	560	**16212.83**§	**16212.83**	$16212.83_{\pm 4.42e-4}$	10.08
VLS22	600	**14641.64**†	14652.28	$14755.90_{\pm 98.88}$	10.08
VLS23	640	**18801.13**§	**18801.13**	$18801.13_{\pm 4.32e-6}$	11.28
VLS24	720	**21389.43**§	**21389.43**	$21389.43_{\pm 7.63e-6}$	13.20
VLS25	760	**17053.26**§	17340.41	$17423.42_{\pm 72.10}$	18.00
VLS26	800	23977.74§	**23977.73**	$23977.73_{\pm 3.49e-5}$	23.76
VLS27	840	**17651.60**†	18326.92	$18364.57_{\pm 37.66}$	26.40
VLS28	880	**26566.04**§	**26566.04**	$26566.04_{\pm 1.33e-6}$	30.00
VLS29	960	**29154.34**§	**29154.34**	$29154.34_{\pm 4.24e-5}$	39.60
VLS30	1040	**31742.64**§	31743.84	$31747.51_{\pm 3.67}$	48.72
VLS31	1120	**34330.94**§	**34330.94**	$34331.54_{\pm 0.60}$	60.00
VLS32	1200	**36919.24**§	37423.94	$37431.73_{\pm 7.79}$	74.88

§ Solution (computed geometrically) [162]; † ORTR [162]

In this implementation of PEGA, migration takes place every 10^4 evaluations; the best individual of the sending population is sent to its neighbor in the ring, and the receiving population always inserts it and removes its worse individual to keep populations at a constant size.

8.2.2 Analysis of Results

In Table 8.4 we show the outcome of executing PEGA on the VLSVRP benchmark; we mark in **boldface** the best result in each one of the 12 instances solved. We show the name of the instance, its size (number of clients), the best solution found in the literature (MSC), the best solution found by PEGA, the average cost found with PEGA, and the average time (in hours) needed by the algorithm.

The values shown in Table 8.4 have been got by averaging over four independent runs for VLS21 and VLS25 (the smallest ones) and over two independent runs for the rest. We are aware that this is a really low number of executions, but the time per execution ranges from 10 hours for the smallest instances to 72 hours for VLS32, which is really a challenge for performing many independent runs on the whole benchmark. This long computational times are motivated by two main reasons: (i) the local search step is harder as the instances grow in size, and (ii) the optimum is actually unknown, and thus our stopping condition must ensure a large enough number of steps to allow the algorithm to converge (hopefully to a better solution compared to the best one reported in literature).

Overall, the results of PEGA are very promising, since it is able of finding the best known solution for 7 out of 12 instances. In fact, PEGA improves the best solution found for VLS26; the numerical difference may seem very small, but the route plan of our solution is clearly new (see Fig. 8.5). We also have

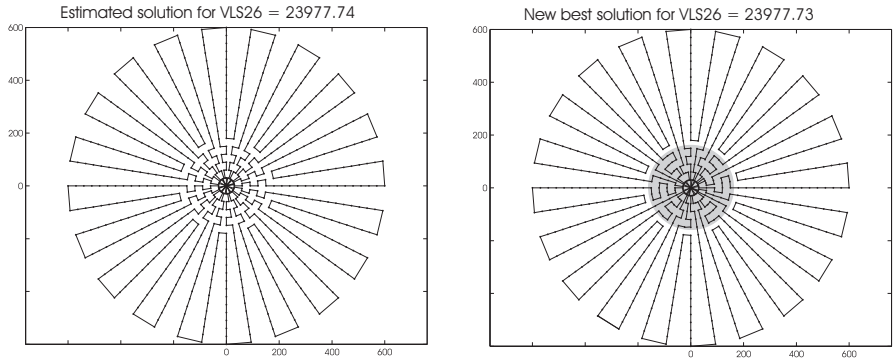


Fig. 8.5. New best solution for VLS26

to remark that the best solution found for all instances (except for VLS22 and VLS27) have been bounded by geometrical properties of the instances, and thus PEGA is the first algorithm able of confirming and maybe improving them in the future.

In Fig. 8.6 we plot a comparison between the best solutions found by PEGA and the three techniques used until now for VLSVRP (three versions of a technique called VRTR [162]). It can be noticed that PEGA gets the overall best results with a few exceptions (VLS25, VLS27 and VLS32). In Table 8.5 we include numerical values representing the percentage of the difference between the results of the mentioned techniques and the best ones found in literature (best performances **boldfaced**). We also show the results of a single cGA using a master-slave technique (i.e., one of the basic components of PEGA) to confirm that it is the overall PEGA strategy the one responsible for the good results. We show average results.

The reader can confirm that PEGA is notably more robust than the other four algorithms, since it is the most accurate technique for 9 out of 12 in-

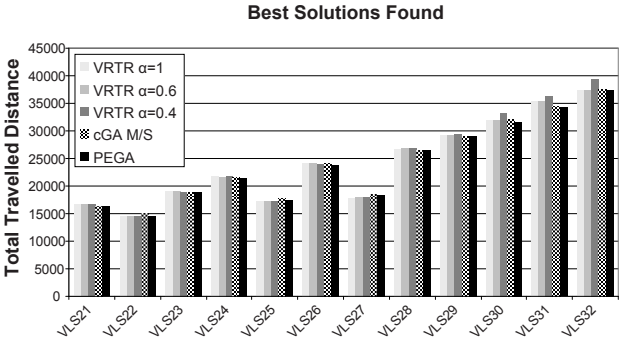


Fig. 8.6. PEGA vs. the previous state of the art results for VLSVRP

Table 8.5. Difference (in %) between the best-so-far solution and PEGA, a cGA MS, and several other state of the art techniques

Instance	VRTR			PEGA	cGA MS
	$\alpha = 1.0$	$\alpha = 0.6$	$\alpha = 0.4$		
VLS21	2.41	2.56	3.25	**0.00**	0.02
VLS22	**0.07**	0.10	0.19	**0.07**	2.17
VLS23	1.09	1.56	0.20	**0.00**	**0.00**
VLS24	1.85	1.06	2.54	**0.00**	$5.61e-3$
VLS25	0.58	0.65	**0.55**	1.68	3.53
VLS26	0.88	0.93	0.13	**0.00**	0.05
VLS27	**0.97**	1.61	1.42	3.83	4.83
VLS28	0.15	0.82	0.83	**0.00**	**0.00**
VLS29	0.09	0.10	0.85	**0.00**	0.02
VLS30	0.74	0.69	4.74	**3.78e − 3**	0.64
VLS31	3.02	2.98	5.85	**0.00**	0.35
VLS32	1.36	**1.33**	6.76	1.37	2.02
Average	1.10	1.20	2.28	**0.58**	1.14

stances. In addition, PEGA finds the best-so-far solution for 7 instances, solutions that were never found by any other metaheuristic, as we mentioned before.

In the last row of Table 8.5 we write the average of the differences between the found solutions and the best-so-far costs. As it can be noticed, PEGA is the best one among the five compared algorithms, even being twice better than the other best technique (VRTR with $\alpha = 1.0$).

8.3 Conclusions

In our quest for more efficient and more accurate algorithms, we have extended the basic concepts of canonical cGAs here in the aim of offering the reader two examples of how these algorithms can be parallelized. Our first proposed model, meta-cGA, is a slight extension of the basic model of cGA using several sub-populations sending their bounding north, south, east, and west individuals after every evaluation to other islands (those located on the north, south, east, and west of the present island cGA). The evaluation of this model (on SAT) states that the parallelism and separate search improve over the more traditional sequential and parallel GAs found in the literature, but with many considerations. The master-slave model, for example, showed its shortcomings here, especially since the evaluations are not too time consuming as one needs to profit from a parallel technique spending time in communications. It is relatively easy to reduce execution times when running distributed algorithms, but it is not that simple to get better numerical results when no collaboration is used or when the distributed model is not powered with problem knowledge. This is what we have confirmed here, since the ba-

sic canonical cGA is a powerful and accurate technique hard to beat (from a numerical point of view) by simple extensions. Additionally, from this work emerges an immediate research line, since the algorithms were only tested on a single problem, and it is really interesting to study if the same results are observed in a larger benchmark of problems.

We have later shown in this chapter a new algorithm (PEGA) for solving very hard instances of CVRP. PEGA has two levels of work: islands distributed in different processors with ProActive in Java, each one running a cGA. To profit from the grid of processors we further use a master-slave model to run low level operators on separate processors. When compared to state-of-the-art techniques, PEGA comes out as a very promising technique, reproducing the existing best bounds for the cost of the solutions for most of the instances, as well as providing new best solutions. The component master-slave cGA has been also evaluated to show that it is the two levels of PEGA who are responsible for its final good results, i.e., the emerging behavior in PEGA is more accurate and interesting than its separate components working alone.

To end this chapter, we must say that parallelism is one of the best ways, not only to reduce the computational time in cGAs and other metaheuristics, but also suited for running decentralized or multilevel searches. All this represents an important and hot line of research able of dealing with optimization challenges that other techniques can only dream of.

9

Designing Cellular Genetic Algorithms for Multi-objective Optimization

I speak Spanish to God, Italian to women, French to men, and German to my horse.

Carlos I de España (1500 - 1558) – World Emperor

Although many research lines deal with benchmarks and complex problem solving, most of them focus in a mono-objective formulation of the target problem. However, it is a well known fact that many real world and combinatorial tasks can be defined in terms of two or more objective functions who are in conflict among them. Thus, it is the formulation, not the problem, what is multi or mono-objective, and the obvious open question is what kind of formulation is more convenient to better solve a problem. Researchers should be aware of the potential benefits of a multi-objective problem (MOP) formulation in terms of efficiency or accuracy for the search of an optimal solution with respect to a mono-objective formulation. In this last, it is not rare to find several terms accounting for penalizations or constraints on a main objective function that often encapsulate the ideas for a more convenient multi-objective formulation.

If we make our choice in favor of a multi-objective formulation for our application, then we will have many advantages; one of these advantages is that of being able of computing more than one equally-good solutions for the same problem (at the same time), what are known as *non-dominated* or *efficient solutions*. Such *Pareto optimal solutions* can be plotted in the space of objective functions, giving raise to a *Pareto front*. Computing the Pareto front for a MOP is the main goal of multi-objective optimization.

Many of the most successful techniques applied today to MOPs are not deterministic, but approximated. Among these last techniques, evolutionary algorithms are one of the most popular, as it happens with NSGA-II [67], PAES [150], and SPEA2 [271]. EAs are good tools for MOPS thanks to their inherent multi-point search, which fits very well the multi-solution needs that designers want from MOP search engines.

In particular, cGAs can be shown to be very effective tools for MOPs. Although many mono-objective approaches exist for cGAs [12, 17] little attention has been paid to their potential multi-objective utilization (which in fact is a common scenario still with many metaheuristics).

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_9, © Springer Science+Business Media, LLC 2008

We briefly describe in this paragraph the main existing approaches of cGAs to the multi-objective (MO) field. In [159] a MO evolution strategy is presented based in a predator-prey idea. The resulting algorithm resembles a cGA, since the solutions (preys) are located in the vertices of a non-directed conex graph, thus implicitly creating a neighborhood where they are hunted by predators. In addition, Murata and Gen presented in [190] an algorithm in which, in order to solve a MOP with n-objectives, the population is structured in an n-dimensional space of weights, and the location of individuals (cells) depends on their own weight vector, which is used to guide the search. In [146] a metapopulation EA (MEA) is discussed; this model is in fact a cellular algorithm in which an extinction process could occur in the population and thus define a region to be colonized by the rest of solutions. Therefore, this model seems to include ideas both from cellular and distributed techniques. Our last example of related works is mainly an application of MO cGAs; in Chap. 14 the algorithm cMOGA [8, 18] is presented, hence representing a clear, complete and first proposal of a multi-objective cellular GA (to our knowledge), used in this case to optimize the broadcasting strategy in ad hoc mobile networks.

In the present chapter we will illustrate the many potential lines of research in MO cGAs by presenting MOCell, an improved version of cMOGA. MOCell is using an external archive (i.e., an external population of solutions) to store the non-dominated solutions computed during the search. This has been pointed out in the literature as an important feature for designing accurate MO algorithms (such as PAES, SPEA2, etc.). In fact, the truly important feature is that of feeding solutions back from the external archive after every iteration into the population used for the search of the algorithm (in the case of MOCell replacing random individuals of its present population of solutions).

For the interested reader, only cMOGA is included in our JCell software, but at this moment we are progressing in incorporating also MOCell in this library. The present version of MOCell is included in a different library of our research group: jMetal [80], which is publicly and freely available at [79].

In this chapter we aim to offer several contributions to this field of MO cGAs. First, we will show an example of how can continuous MOPs be solved with this technique in which an external archive for storing and feedback is used. We also will evaluate MOCell either on restricted and unrestricted benchmarks, comparing the results against the state of the art in MOPs: NSGA-II and SPEA2. Finally, we will discuss the importance of the replacement and feedback strategies in the algorithm and analyze its results versus the mentioned NSGA-II and SPEA2.

The rest of the chapter is organized in the following manner. In Sect. 9.1 we explain several basic though important concepts for multi-objective optimization. In Sect. 9.2 we describe the internals of MOCell and investigate several design decisions. To show the potentials of these ideas we show some results after an extensive experimentation in Sect. 9.3. To conclude this chapter, we make a global discussion of the involved issues in Sect. 9.4.

9.1 Background on Multi-objective Optimization

We present in this section some of the most basic concepts on multi-objective optimization to get the reader familiar with the concepts. Indeed, we will define MOP ideas, Pareto optimality, Pareto dominance, optimal Pareto set, and Pareto front. We will assume minimization of all the objectives (without losing generality in our statements).

A multi-objective problem (MOP) can be defined in the following manner:

Definition 9.1 (MOP). *Find a vector* $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ *satisfying* m *inequality restrictions* $g_i(\mathbf{x}) \geq 0, i = 1, 2, \ldots, m$, p *equality restrictions* $h_i(\mathbf{x}) = 0, i = 1, 2, \ldots, p$, *and minimizing the function vector* $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T$, *where* $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ *is the vector of decision variables.* ■

The set of all values satisfying the restrictions defines the *feasibility region* Ω, and any point in $\mathbf{x} \in \Omega$ is said to be a *feasible solution.*

As previously mentioned, in this problem we look for many solutions: *Pareto optima* or *efficient solutions*, whose definition follows:

Definition 9.2 (Pareto Optimality). *One point* $\mathbf{x}^* \in \Omega$ *is a Pareto optimum if, for every* $\mathbf{x} \in \Omega$ *and* $I = \{1, 2, \ldots, k\}$, *or else* $\forall_{i \in I} (f_i(\mathbf{x}) = f_i(\mathbf{x}^*))$, *or at least there is a* $i \in I \mid f_i(\mathbf{x}) > f_i(\mathbf{x}^*)$. ■

This definition reveals $\mathbf{x}^*$ as a Pareto optimum if no other feasible vector exists $\mathbf{x}$ in which some objective is improved over it, unless it implies the worsening of at least one other different objective function. An additional important and related definition is given below:

Definition 9.3 (Pareto Dominance). *A vector* $\mathbf{u} = (u_1, \ldots, u_k)$ *is said to dominate another vector* $\mathbf{v} = (v_1, \ldots, v_k)$ *(represented by* $\mathbf{u} \preceq \mathbf{v}$*) iff* $\mathbf{u}$ *is partially smaller than* $\mathbf{v}$, *that is,* $\forall i \in \{1, \ldots, k\}$, $u_i \leq v_i \wedge \exists\, i \in \{1, \ldots, k\} : u_i < v_i$. ■

Definition 9.4 (Pareto Optimal Set). *For a given MOP* $\mathbf{f}(\mathbf{x})$, *the Pareto optimal set is defined as* $\mathcal{P}^* = \{\mathbf{x} \in \Omega | \neg \exists \mathbf{x'} \in \Omega, \mathbf{f}(\mathbf{x'}) \preceq \mathbf{f}(\mathbf{x})\}$. ■

Definition 9.5 (Pareto Front). *For a given MOP* $\mathbf{f}(\mathbf{x})$ *and its corresponding Pareto optimal set* $\mathcal{P}^*$, *the Pareto front is defined by* $\mathcal{PF}^* = \{\mathbf{f}(\mathbf{x}), \mathbf{x} \in \mathcal{P}^*\}$. ■

Obtaining the Pareto front for a MOP is the main goal of multi-objective optimization. In practice, since a Pareto front is a finite set of points, a practical good solution for a MOP should contain points as well spread and as close as possible with respect to the optimal Pareto front. The aim is to better serve for the subsequent decision making phase, usually performed by an expert in the application.

Algorithm 9.1 Pseudo-code of MOCell

```
1. proc Evolve(mocell)          //Parameters in 'mocell'
2. Pareto_front = Create_PFront() //Create an empty Pareto front
3. while !StopCondition() do
4.    for individual ← 1 until mocell.popSize do
5.       neighbors ← GetNeighbors(mocell,position(individual));
6.       parents ← Select(neighbors);
7.       offspring ← Recombination(mocell.Pc,parents);
8.       offspring ← Mutation(mocell.Pm,offspring);
9.       Evaluate(offspring);
10.      Insert(position(individual),offspring,mocell,aux_population);
11.      InsertInParetoFront(individual, Pareto_front);
12.   end for
13.   mocell.population ← aux_population;
14.   mocell.population ← Feedback(mocell,Pareto_Front);
15. end while
16. end proc Evolve;
```

9.2 The MOCell Algorithm

We here present MOCell [193], one of our own proposals for multi-objective optimization using a cGA as the basic template for the search. The reader can find its pseudo-code in Alg. 9.1. We can observe there the basics of a regular cGA (see Alg. 1.2 for the details). The main difference in MOCell is the inclusion of the notion of Pareto front. In our proposal, this Pareto front appears as an additional finite population made up of a set of non-dominated solutions. To deal with the insertion of solutions into the Pareto front (in order to improve its diversity) MOCell uses a density estimator based in a mechanism named *crowding* proposed for the well-known NSGA-II algorithm [67]. This method is also used as a tool for removing solutions from the archive whenever it reaches its maximum size during the search.

MOCell starts working an empty Pareto front (line 2 in Alg. 9.1). Individuals are located in a toroidal 2-dimensional mesh; they undergo the reproductive cycle (lines 4 to 12) until the stopping condition is met (line 3). In consequence, for every individual, the algorithm selects two parents in its neighborhood, recombine them to create one offspring who is later mutated, evaluated and inserted back either into an auxiliary population (only if it is not dominated by the presently considered individual) and also into the Pareto front. Finally, the present population is replaced in a single step by the auxiliary population (line 13, i.e., it is a synchronous cGA). Additionally, a feedback process is executed in order to randomly select a given number of individuals of the population grid and replace them with the best solutions coming from the archive (line 14).

Table 9.1. Parameters used for the evaluation of MOCell

Population size	100 individuals (10×10)
Stop condition	25,000 evaluations of the objective function
Neighborhood	C9
Parent selection	Binary tournament + binary tournament
Recombination	SBX, $p_c = 1.0$
Mutation	Polynomial, $p_m = 1.0/L$ (L = Individual length)
Replacement	Repl_if_Better (NSGA-II Crowding)
Size of the archive	100 individuals
Density estimator	Crowding distance
Feedback	20 individuals

MOCell is also using a mechanism to deal with restrictions for those problems having such feature. This operation is the same as the one used in NSGA-II, and it induces a partial order in which the individual violating a lower number or restrictions is considered the best. In case of equal number of violated restrictions the individuals are compared attending to their fitness.

In Table 9.1 we show the parameters used for the evaluation of MOCell in this chapter. We have selected a square toroidal grid of 100 individuals as the base population. The neighborhood used is C9 (compact nine), in which 9 individuals are selected around the present point of computations by using all the immediate neighbor individuals located at one hop of distance in vertical, horizontal and diagonal directions. Since we are dealing with continuous problems, we use the SBX recombination, a very well-known one, and the mutation applies a polynomial operation in the standard way in which NSGA-II and SPEA2 apply it. The application rates for these two operators are $p_c = 1.0$ and $p_m = 1/L$, respectively.

The offspring will replace the presently considered individual only if it is better. However, in multi-objective optimization we need to define this idea of a better solution more precisely since we have many objectives. Our approach is to replace the present individual if it is dominated by the new offspring, or if they two are mutually non-dominated but the existing individual is having a worse crowding distance (as defined in NSGA-II) in a mini-population made of the whole present neighborhood and the offspring. In order to insert the individual in the Pareto front of the archive, it is also ranked according to the crowding distance metric. Therefore, when a non-dominated solution is going to be inserted and the archive is full, the archived solution having the worse crowding distance is removed. Finally, after every iteration of the algorithm, 20 randomly chosen individuals from the population are replaced by the 20 best solutions (using crowding) of the external archive. This feedback is very important for the success of the algorithm, since it promotes the search.

9.2.1 Extensions to MOCell

In our quest of new algorithms able of unseen performances we are going to further consider in this section several extensions to the basic MOCell presented before. The justification is clear, since some of the parameters regulating the behavior of the algorithm need additional tuning and investigation in the aim of upgrading the efficiency and especially the accuracy of MOCell.

Therefore, we are going to focus our analysis towards the updating policy used in the population grid, the policy for performing the feedback from the archive, and the replacement technique used. We will discuss a total of 5 new algorithms inspired in MOCell but using new tools for the evolution of solutions. Some of these solutions will take us far from an orthodox definition of a cGA, but our goal is to better solve problems and offer new ideas, not to stick to the same search template as a dogma. Let us discuss these novelties in the structure of MOCell:

- **Synchronism.** The previously studied algorithm is updating all the individuals in the grid at once, i.e., it is synchronous. Now we will deal with asynchronous updating policies (see again Chap. 1); since there are many of them we will analyze here only Line Sweep (LS), in which the population grid is scanned from top left, line by line, and using for each individual the recent solutions computed in its neighborhood instead of storing them in an auxiliary population.
- **Feedback.** In MOCell, selected individuals from the archive replace some randomly chosen individuals in the population grid. This behavior shows some constraints, since this blind feedback can lead to lose good individuals (fitness or diversity) from the population. Also, after several of these explicit feedback steps the population itself could disappear in favor of mere copies of the contents of the archive. Thus, we seek for an implicit feedback executed during selection by using as one of the parents in every step some solution from the archive, while the other parent is still selected from the presently considered neighborhood.
- **Replacement.** Instead of replacing the presently considered individual, we will consider its whole present neighborhood, and effectively replace the worst individual living in it.

With all these ideas in mind six new extensions of MOCell come to scene (their differences are summarized in Table 9.2):

Table 9.2. Features of the proposed extensions to MOCell

Algorithm	Synchronicity	Implicit feedback	Global replacement
sMOCell1	Synch.	No	No
sMOCell2	Synch.	Yes	No
aMOCell1	Asynch.	No	No
sMOCell2	Asynch.	Yes	No
sMOCell3	Asynch.	No	Yes
sMOCell4	Asynch.	Yes	Yes

- **sMOCell1**: The MOCell proposal described in Alg. 9.1.
- **sMOCell2**: The basic MOCell in which no generational replacement from the archive is used, but instead the implicit feedback through selection of the parents is applied.
- **aMOCell1**: Asynchronous extension of sMOCell1.
- **aMOCell2**: aMOCell1 + feedback through parental selection.
- **aMOCell3**: aMOCell1 + replacement of the worst individual in its neighborhood.
- **aMOCell4**: A Combination of aMOCell2 and aMOCell3.

In next section we present and discuss all these extensions, which will be evaluated in detail.

9.3 Experimental Analysis

Let us address here the evaluation of the above explained six extensions of MOCell. To this end, we will use metrics such as GD (to measure convergence), and Δ (to measure diversity); in addition, we will use *hypervolume* –HV– (please refer to Chap. 5 for more details on metrics). For our analysis several standard benchmark problems are used, such as WFG, and functions ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6. All them are described in detail in Appendix A.

In Tables 9.3 to 9.5 we show the numerical results of all six extensions for the three metrics. We include the median ($\tilde{x}$) and the interquartile index (IQR) of the results after 100 independent runs. The best value for all the algorithms (for a given metric and problem) is shown in **boldface**. The statistical analysis is of course very important to sustain the final claims; first, we apply the Kolmogorov-Smirnov test on the data to check their normality; if they are normally distributed then an ANOVA test is applied; otherwise we will apply a Kruskal-Wallis test. As in the rest of this book, the confidence level is 95%. Indeed, a '+' symbol means statistical significance (i.e., p-value smaller than 0.05), in Tables 9.3 to 9.5.

In Table 9.3 we show the actual values for the GD metric. From this point of view, aMOCell4 gets the best results (lowest values) in 6 out of the 14 problems. In 5 of them the statistical confidence exists ("+" in the last column). As to the influence of the updating policy, the asynchronous extensions compute a Pareto front which is nearer to the optimal Pareto front with respect to their synchronous counterparts in 11 out of the 14 problems.

Let us now turn to the results concerning the Δ metric in Table 9.4. Again, aMOCell4 excels in 6 out of the 14 studied MOPs. Also, the asynchronous extensions are the best under this dispersion metric in 13 out of the 14 average Pareto Fronts.

Finally, as to the Hypervolume metric HV, we can notice in Table 9.5 that it provides again the same conclusions and therefore supports the same claims. Firstly, aMOCell4 is the best (larger values) in 8 out of 14 MOPs; secondly, the asynchronous extensions are the best in all the problems (except for WFG6).

Table 9.3. Comparison among the different extensions of MOCell: median and interquartile range of the metric *GD*

MOP	sMOCell1	sMOCell2	aMOCell1	aMOCell2	aMOCell3	aMOCell4	Test
	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	
ZDT1	$6.288e-4$ $1.5e-4$	$2.749e-4$ $7.5e-5$	$4.207e-4$ $7.2e-5$	$2.518e-4$ $4.5e-5$	$2.222e-4$ $4.1e-5$	$\mathbf{1.753e-4}$ $\mathbf{2.0e-5}$	+
ZDT2	$5.651e-4$ $2.0e-4$	$1.778e-4$ $1.1e-4$	$2.884e-4$ $1.9e-4$	$1.111e-4$ $1.1e-4$	$1.197e-4$ $5.3e-5$	$\mathbf{5.629e-5}$ $\mathbf{2.5e-5}$	+
ZDT3	$3.326e-4$ $8.5e-5$	$2.493e-4$ $3.0e-5$	$2.644e-4$ $4.4e-5$	$2.427e-4$ $2.8e-5$	$2.077e-4$ $2.0e-5$	$\mathbf{2.008e-4}$ $\mathbf{1.8e-5}$	+
ZDT4	$9.668e-4$ $6.4e-4$	$3.848e-4$ $2.9e-4$	$7.847e-4$ $5.9e-4$	$4.235e-4$ $3.3e-4$	$6.179e-4$ $4.0e-4$	$\mathbf{3.293e-4}$ $\mathbf{2.0e-4}$	+
ZDT6	$3.963e-3$ $1.3e-3$	$1.080e-3$ $2.0e-4$	$2.397e-3$ $7.8e-4$	$9.334e-4$ $1.3e-4$	$8.778e-4$ $1.3e-4$	$\mathbf{6.323e-4}$ $\mathbf{3.4e-5}$	+
WFG1	$1.962e-4$ $8.0e-3$	$\mathbf{1.859e-4}$ $\mathbf{1.9e-5}$	$1.906e-4$ $6.5e-3$	$1.889e-4$ $2.0e-5$	$1.921e-4$ $8.1e-3$	$2.052e-4$ $1.0e-2$	+
WFG2	$4.408e-4$ $1.4e-4$	$4.339e-4$ $1.2e-4$	$4.410e-4$ $1.3e-4$	$\mathbf{4.316e-4}$ $\mathbf{1.3e-4}$	$4.337e-4$ $1.3e-4$	$4.336e-4$ $7.1e-5$	+
WFG3	$1.372e-4$ $1.4e-5$	$1.349e-4$ $1.5e-5$	$1.375e-4$ $1.8e-5$	$\mathbf{1.340e-4}$ $\mathbf{1.3e-5}$	$1.367e-4$ $1.5e-5$	$1.354e-4$ $1.4e-5$	+
WFG4	$6.423e-4$ $2.2e-5$	$6.259e-4$ $2.6e-5$	$6.396e-4$ $2.6e-5$	$\mathbf{6.252e-4}$ $\mathbf{2.4e-5}$	$6.341e-4$ $2.6e-5$	$6.253e-4$ $3.2e-5$	+
WFG5	$2.634e-3$ $2.6e-5$	$2.633e-3$ $1.4e-5$	$2.636e-3$ $3.4e-5$	$\mathbf{2.631e-3}$ $\mathbf{1.4e-5}$	$2.633e-3$ $1.2e-5$	$2.635e-3$ $1.1e-5$	+
WFG6	$\mathbf{4.984e-4}$ $\mathbf{4.3e-4}$	$1.210e-3$ $2.1e-3$	$5.146e-4$ $7.1e-4$	$1.268e-3$ $3.4e-3$	$5.976e-4$ $7.1e-4$	$1.906e-3$ $3.4e-3$	+
WFG7	$3.069e-4$ $2.2e-5$	$3.048e-4$ $2.3e-5$	$3.025e-4$ $2.1e-5$	$3.038e-4$ $2.7e-5$	$3.067e-4$ $2.4e-5$	$\mathbf{3.011e-4}$ $\mathbf{2.4e-5}$	•
WFG8	$1.009e-2$ $6.6e-3$	$1.460e-2$ $5.4e-3$	$\mathbf{1.000e-2}$ $\mathbf{6.0e-3}$	$1.468e-2$ $3.3e-3$	$1.434e-2$ $5.2e-3$	$1.474e-2$ $4.9e-3$	+
WFG9	$1.072e-3$ $6.1e-5$	$\mathbf{1.055e-3}$ $\mathbf{5.3e-5}$	$1.081e-3$ $5.5e-5$	$1.067e-3$ $6.6e-5$	$1.067e-3$ $5.8e-5$	$1.065e-3$ $6.0e-5$	+

Table 9.4. Comparison among the different extensions of MOCell: median and interquartile range of the metric Δ

MOP	sMOCell1	sMOCell2	aMOCell1	aMOCell2	aMOCell3	aMOCell4	Test
	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	
ZDT1	$1.541e-1$ $2.1e-2$	$9.645e-2$ $1.4e-2$	$1.345e-1$ $1.9e-2$	$9.161e-2$ $1.3e-2$	$1.011e-1$ $1.7e-2$	$\mathbf{7.493e-2}$ $\mathbf{1.3e-2}$	+
ZDT2	$1.753e-1$ $3.8e-2$	$9.907e-2$ $1.9e-2$	$1.363e-1$ $3.5e-2$	$9.089e-2$ $2.4e-2$	$1.003e-1$ $2.1e-2$	$\mathbf{8.095e-2}$ $\mathbf{1.3e-2}$	+
ZDT3	$7.106e-1$ $7.5e-3$	$7.073e-1$ $7.4e-3$	$7.091e-1$ $7.8e-3$	$7.069e-1$ $7.0e-3$	$\mathbf{7.039e-1}$ $\mathbf{4.0e-3}$	$7.054e-1$ $5.4e-3$	+
ZDT4	$1.964e-1$ $9.1e-2$	$1.257e-1$ $3.6e-2$	$1.854e-1$ $6.3e-2$	$1.324e-1$ $4.5e-2$	$1.419e-1$ $3.1e-2$	$\mathbf{1.089e-1}$ $\mathbf{2.5e-2}$	+
ZDT6	$3.806e-1$ $1.1e-1$	$1.513e-1$ $2.5e-2$	$2.953e-1$ $7.3e-2$	$1.363e-1$ $1.9e-2$	$1.536e-1$ $1.8e-2$	$\mathbf{9.234e-2}$ $\mathbf{1.1e-2}$	+
WFG1	$5.469e-1$ $9.3e-2$	$5.653e-1$ $7.6e-2$	$5.298e-1$ $1.0e-1$	$5.571e-1$ $7.3e-2$	$\mathbf{4.679e-1}$ $\mathbf{1.2e-1}$	$5.790e-1$ $8.6e-2$	+
WFG2	$7.490e-1$ $1.1e-2$	$\mathbf{7.468e-1}$ $\mathbf{1.0e-2}$	$7.474e-1$ $1.1e-2$	$7.468e-1$ $9.9e-3$	$7.468e-1$ $1.0e-2$	$7.471e-1$ $8.5e-3$	+
WFG3	$3.698e-1$ $9.8e-3$	$3.657e-1$ $8.0e-3$	$3.725e-1$ $8.2e-3$	$\mathbf{3.634e-1}$ $\mathbf{7.3e-3}$	$3.684e-1$ $7.2e-3$	$3.648e-1$ $8.7e-3$	+
WFG4	$1.349e-1$ $1.9e-2$	$1.341e-1$ $1.7e-2$	$1.335e-1$ $1.7e-2$	$1.336e-1$ $1.9e-2$	$1.335e-1$ $1.7e-2$	$\mathbf{1.333e-1}$ $\mathbf{1.7e-2}$	•
WFG5	$1.311e-1$ $2.5e-2$	$1.298e-1$ $1.7e-2$	$1.377e-1$ $2.3e-2$	$\mathbf{1.289e-1}$ $\mathbf{2.3e-2}$	$1.300e-1$ $2.3e-2$	$1.293e-1$ $1.8e-2$	+
WFG6	$1.178e-1$ $2.1e-2$	$1.339e-1$ $3.4e-2$	$\mathbf{1.167e-1}$ $\mathbf{2.7e-2}$	$1.344e-1$ $4.6e-2$	$1.190e-1$ $2.7e-2$	$1.348e-1$ $4.1e-2$	+
WFG7	$1.059e-1$ $1.8e-2$	$1.096e-1$ $1.7e-2$	$\mathbf{1.033e-1}$ $\mathbf{1.6e-2}$	$1.069e-1$ $2.1e-2$	$1.040e-1$ $1.7e-2$	$1.084e-1$ $2.1e-2$	+
WFG8	$5.596e-1$ $6.3e-2$	$5.664e-1$ $8.4e-2$	$5.710e-1$ $7.2e-2$	$5.691e-1$ $5.0e-2$	$\mathbf{5.531e-1}$ $\mathbf{6.4e-2}$	$5.703e-1$ $6.7e-2$	+
WFG9	$1.597e-1$ $1.8e-2$	$1.449e-1$ $1.8e-2$	$1.609e-1$ $2.1e-2$	$1.482e-1$ $1.8e-2$	$1.606e-1$ $1.8e-2$	$\mathbf{1.435e-1}$ $\mathbf{1.7e-2}$	+

Table 9.5. Comparison among the different extensions of MOCell: median and interquartile range of the metric HV

MOP	sMOCell1	sMOCell2	aMOCell1	aMOCell2	aMOCell3	aMOCell4	Test
	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	$\tilde{x}_{IQR}$	
ZDT1	$6.543e-1$ $2.0e-3$	$6.592e-1$ $7.3e-4$	$6.573e-1$ $1.1e-3$	$6.595e-1$ $7.3e-4$	$6.603e-1$ $5.2e-4$	**$6.610e-1$** **$2.7e-4$**	+
ZDT2	$3.216e-1$ $2.8e-3$	$3.265e-1$ $1.6e-3$	$3.256e-1$ $2.6e-3$	$3.274e-1$ $1.7e-3$	$3.276e-1$ $8.1e-4$	**$3.284e-1$** **$5.1e-4$**	+
ZDT3	$5.111e-1$ $2.2e-3$	$5.135e-1$ $8.2e-4$	$5.132e-1$ $1.2e-3$	$5.137e-1$ $8.1e-4$	**$5.152e-1$** **$2.8e-4$**	$5.152e-1$ $4.0e-4$	+
ZDT4	$6.487e-1$ $9.6e-3$	$6.573e-1$ $4.3e-3$	$6.517e-1$ $8.4e-3$	$6.568e-1$ $4.5e-3$	$6.539e-1$ $5.9e-3$	**$6.580e-1$** **$3.2e-3$**	+
ZDT6	$3.487e-1$ $1.7e-2$	$3.885e-1$ $3.1e-3$	$3.699e-1$ $1.0e-2$	$3.909e-1$ $2.0e-3$	$3.920e-1$ $2.4e-3$	**$3.970e-1$** **$8.4e-4$**	+
WFG1	$5.491e-1$ $1.1e-1$	$6.047e-1$ $5.8e-2$	$5.906e-1$ $1.2e-1$	$5.983e-1$ $1.0e-1$	**$6.115e-1$** **$1.2e-1$**	$5.043e-1$ $1.7e-1$	+
WFG2	$5.616e-1$ $2.9e-3$	$5.616e-1$ $2.7e-3$	**$5.616e-1$** **$2.8e-3$**	$5.616e-1$ $2.7e-3$	$5.616e-1$ $2.7e-3$	5.616e-1 $1.1e-3$	•
WFG3	$4.420e-1$ $2.0e-4$	$4.420e-1$ $1.6e-4$	$4.420e-1$ $3.0e-4$	**$4.420e-1$** **$1.6e-4$**	$4.420e-1$ $2.5e-4$	$4.420e-1$ $1.6e-4$	+
WFG4	$2.187e-1$ $3.1e-4$	$2.186e-1$ $3.2e-4$	$2.186e-1$ $2.8e-4$	$2.186e-1$ $2.9e-4$	$2.187e-1$ $2.9e-4$	**$2.188e-1$** **$2.6e-4$**	+
WFG5	$1.961e-1$ $7.5e-5$	$1.962e-1$ $5.4e-5$	$1.961e-1$ $7.5e-5$	$1.962e-1$ $6.9e-5$	$1.962e-1$ $7.4e-5$	**$1.962e-1$** **$4.7e-5$**	+
WFG6	**$2.051e-1$** **$7.0e-3$**	$1.949e-1$ $2.8e-2$	$2.049e-1$ $1.1e-2$	$1.940e-1$ $4.3e-2$	$2.036e-1$ $1.1e-2$	$1.859e-1$ $4.2e-2$	+
WFG7	$2.104e-1$ $1.7e-4$	$2.104e-1$ $1.7e-4$	$2.104e-1$ $1.6e-4$	$2.104e-1$ $2.0e-4$	$2.104e-1$ $2.0e-4$	**$2.105e-1$** **$1.6e-4$**	+
WFG8	$1.456e-1$ $2.1e-2$	$1.472e-1$ $3.0e-3$	$1.459e-1$ $4.9e-3$	$1.466e-1$ $2.5e-3$	$1.462e-1$ $2.8e-3$	**$1.479e-1$** **$2.8e-3$**	+
WFG9	$2.380e-1$ $2.2e-3$	$2.389e-1$ $2.4e-3$	$2.375e-1$ $3.1e-3$	**$2.390e-1$** **$2.0e-3$**	$2.380e-1$ $2.3e-3$	$2.381e-1$ $3.6e-3$	+

If we focus now on the replacement policy, it is clear that the new policy (one parent from the archive) is better than an explicit feedback (as it happens in the canonical MOCell). This fact is obvious both for the synchronous extensions (sMOCell2 performs better than sMOCell1 on problems 12, 10, and 9) and for the asynchronous ones (aMOCell2 outperforms aMOCell1 in 11, 10, and 11 problems according to metrics GD, Δ, and HV, respectively).

The reader is maybe thinking that the differences among all the extensions are too small, but this is only apparently, and due to the normalization process we are using before the calculation of the metrics to be able of such a large comparison. In fact, most of the results are statistically significant. As a conclusion, aMOCell4 is the best extension of MOCell thanks to the combined effect of the new replacement and feedback strategies.

Since we are also interested in comparing to the rest of popular algorithms from the literature we will compare aMOCell4 against NSGA-II and SPEA-2. These two algorithms have been implemented in the same framework as aMOCell4 in order to reinforce the fairness of the process and avoid well-known bugs in some of the existing implementations available in Internet. In [193] we have already shown that the basic MOCell is competitive with respect to their available implementations.

In Tables 9.6, 9.7, and 9.8 we include the results of applying aMOCell4, NSGA-II, and SPEA2 to our benchmark for the mentioned metrics: GD, Δ, and HV, respectively. If we start our analysis by the accuracy (distance to the optimal Pareto front) we can notice in Table 9.6 that the cellular based approach gets the best (smallest) values for 10 problems, out of the 14 used

Table 9.6. Comparison among aMOCell4, NSGA-II and SPEA2. Median and interquartile range of the metric *GD*

MOP	aMOCell4 $\tilde{x}_{IQR}$	NSGA-II $\tilde{x}_{IQR}$	SPEA2 $\tilde{x}_{IQR}$	Test
ZDT1	$\mathbf{1.753e-4}_{\mathbf{2.0e-5}}$	$2.198e-4_{4.8e-5}$	$2.211e-4_{2.8e-5}$	+
ZDT2	$\mathbf{5.629e-5}_{\mathbf{2.5e-5}}$	$1.674e-4_{4.3e-5}$	$1.770e-4_{4.8e-5}$	+
ZDT3	$\mathbf{2.008e-4}_{\mathbf{1.8e-5}}$	$2.126e-4_{2.1e-5}$	$2.320e-4_{2.0e-5}$	+
ZDT4	$\mathbf{3.293e-4}_{\mathbf{2.0e-4}}$	$4.353e-4_{3.2e-4}$	$5.753e-4_{4.4e-4}$	+
ZDT6	$\mathbf{6.323e-4}_{\mathbf{3.4e-5}}$	$1.010e-3_{1.3e-4}$	$1.750e-3_{2.9e-4}$	+
WFG1	$2.052e-4_{1.0e-2}$	$\mathbf{1.967e-4}_{\mathbf{8.3e-3}}$	$6.438e-4_{1.0e-2}$	+
WFG2	$\mathbf{4.336e-4}_{\mathbf{7.1e-5}}$	$5.196e-4_{1.7e-4}$	$4.474e-4_{1.2e-4}$	+
WFG3	$\mathbf{1.354e-4}_{\mathbf{1.4e-5}}$	$1.553e-4_{1.9e-5}$	$1.448e-4_{1.2e-5}$	+
WFG4	$\mathbf{6.253e-4}_{\mathbf{3.2e-5}}$	$6.870e-4_{1.4e-4}$	$6.377e-4_{3.0e-5}$	+
WFG5	$\mathbf{2.635e-3}_{\mathbf{1.1e-5}}$	$2.655e-3_{3.2e-5}$	$2.718e-3_{1.7e-5}$	+
WFG6	$1.906e-3_{3.4e-3}$	$5.539e-4_{6.5e-4}$	$\mathbf{4.654e-4}_{\mathbf{6.7e-4}}$	+
WFG7	$\mathbf{3.011e-4}_{\mathbf{2.4e-5}}$	$3.444e-4_{4.7e-5}$	$3.020e-4_{4.5e-5}$	+
WFG8	$1.474e-2_{4.9e-3}$	$\mathbf{1.446e-2}_{\mathbf{5.3e-3}}$	$1.569e-2_{6.0e-3}$	+
WFG9	$1.065e-3_{6.0e-5}$	$1.223e-3_{2.1e-4}$	$\mathbf{9.313e-4}_{\mathbf{9.1e-5}}$	+

Table 9.7. Comparison among aMOCell4, NSGA-II and SPEA2. Median and interquartile range of the metric Δ

MOP	aMOCell4 $\tilde{x}_{IQR}$	NSGA-II $\tilde{x}_{IQR}$	SPEA2 $\tilde{x}_{IQR}$	Test
ZDT1	$\mathbf{7.493e-2}_{\mathbf{1.3e-2}}$	$3.753e-1_{4.2e-2}$	$1.486e-1_{1.8e-2}$	+
ZDT2	$\mathbf{8.095e-2}_{\mathbf{1.3e-2}}$	$3.814e-1_{3.9e-2}$	$1.558e-1_{2.8e-2}$	+
ZDT3	$\mathbf{7.054e-1}_{\mathbf{5.4e-3}}$	$7.458e-1_{2.0e-2}$	$7.099e-1_{7.7e-3}$	+
ZDT4	$\mathbf{1.089e-1}_{\mathbf{2.5e-2}}$	$3.849e-1_{5.3e-2}$	$2.612e-1_{1.7e-1}$	+
ZDT6	$\mathbf{9.234e-2}_{\mathbf{1.1e-2}}$	$3.591e-1_{4.6e-2}$	$2.268e-1_{3.0e-2}$	+
WFG1	$\mathbf{5.790e-1}_{\mathbf{8.6e-2}}$	$7.170e-1_{4.5e-2}$	$6.578e-1_{7.0e-2}$	+
WFG2	$\mathbf{7.471e-1}_{\mathbf{8.5e-3}}$	$7.968e-1_{1.5e-2}$	$7.519e-1_{1.1e-2}$	+
WFG3	$\mathbf{3.648e-1}_{\mathbf{8.7e-3}}$	$6.101e-1_{3.8e-2}$	$4.379e-1_{1.3e-2}$	+
WFG4	$\mathbf{1.333e-1}_{\mathbf{1.7e-2}}$	$3.835e-1_{4.3e-2}$	$2.681e-1_{3.1e-2}$	+
WFG5	$\mathbf{1.293e-1}_{\mathbf{1.8e-2}}$	$4.077e-1_{4.0e-2}$	$2.805e-1_{2.7e-2}$	+
WFG6	$\mathbf{1.348e-1}_{\mathbf{4.1e-2}}$	$3.807e-1_{4.2e-2}$	$2.506e-1_{2.4e-2}$	+
WFG7	$\mathbf{1.084e-1}_{\mathbf{2.1e-2}}$	$3.836e-1_{4.4e-2}$	$2.453e-1_{2.7e-2}$	+
WFG8	$\mathbf{5.703e-1}_{\mathbf{6.7e-2}}$	$6.472e-1_{5.1e-2}$	$6.108e-1_{5.7e-2}$	+
WFG9	$\mathbf{1.435e-1}_{\mathbf{1.7e-2}}$	$3.994e-1_{3.9e-2}$	$2.945e-1_{2.4e-2}$	+

Table 9.8. Comparison among aMOCell4, NSGA-II and SPEA2. Median and interquartile range of the metric *HV*

MOP	aMOCell4 $\tilde{x}_{IQR}$	NSGA-II $\tilde{x}_{IQR}$	SPEA2 $\tilde{x}_{IQR}$	Test
ZDT1	$\mathbf{6.610e-1}_{\mathbf{2.7e-4}}$	$6.594e-1_{4.0e-4}$	$6.600e-1_{3.5e-4}$	+
ZDT2	$\mathbf{3.284e-1}_{\mathbf{5.1e-4}}$	$3.261e-1_{4.8e-4}$	$3.263e-1_{7.4e-4}$	+
ZDT3	$\mathbf{5.152e-1}_{\mathbf{4.0e-4}}$	$5.148e-1_{2.7e-4}$	$5.141e-1_{3.4e-4}$	+
ZDT4	$\mathbf{6.580e-1}_{\mathbf{3.2e-3}}$	$6.552e-1_{4.7e-3}$	$6.518e-1_{1.0e-2}$	+
ZDT6	$\mathbf{3.970e-1}_{\mathbf{8.4e-4}}$	$3.887e-1_{2.2e-3}$	$3.785e-1_{4.3e-3}$	+
WFG1	$5.043e-1_{1.7e-1}$	$\mathbf{5.140e-1}_{\mathbf{1.5e-1}}$	$4.337e-1_{1.4e-1}$	+
WFG2	$5.616e-1_{1.1e-3}$	$\mathbf{5.631e-1}_{\mathbf{2.9e-3}}$	$5.615e-1_{2.9e-3}$	•
WFG3	$\mathbf{4.420e-1}_{\mathbf{1.6e-4}}$	$4.411e-1_{3.2e-4}$	$4.418e-1_{2.2e-4}$	+
WFG4	$\mathbf{2.188e-1}_{\mathbf{2.6e-4}}$	$2.173e-1_{5.3e-4}$	$2.181e-1_{3.4e-4}$	+
WFG5	$\mathbf{1.962e-1}_{\mathbf{4.7e-5}}$	$1.948e-1_{4.8e-4}$	$1.956e-1_{1.5e-4}$	+
WFG6	$1.859e-1_{4.2e-2}$	$2.033e-1_{9.9e-3}$	$\mathbf{2.056e-1}_{\mathbf{1.1e-2}}$	+
WFG7	$\mathbf{2.105e-1}_{\mathbf{1.6e-4}}$	$2.088e-1_{4.3e-4}$	$2.098e-1_{2.7e-4}$	+
WFG8	$\mathbf{1.479e-1}_{\mathbf{2.8e-3}}$	$1.470e-1_{2.3e-3}$	$1.469e-1_{1.7e-3}$	+
WFG9	$2.381e-1_{3.6e-3}$	$2.372e-1_{2.2e-3}$	$\mathbf{2.386e-1}_{\mathbf{2.2e-3}}$	+

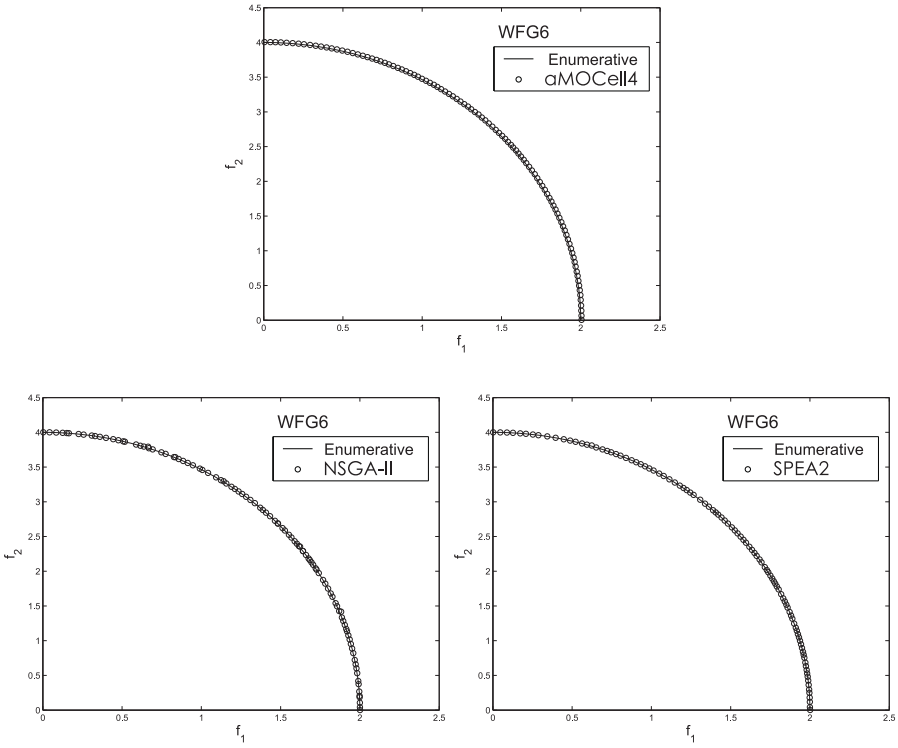


Fig. 9.1. Best found fronts for aMOCell4, NSGA-II, and SPEA2, all solving WFG6

in the benchmark. The mentioned aMOCell4 is specially accurate for all the ZDT problem family. As to the WFG family, NSGA-II and SPEA2 are only the most accurate techniques for 4 problems (out of 9), while aMOCell4 is able of outperforming them in the rest of 5 MOPs. The conclusion then is that aMOCell4 outperforms the two standard procedures in multi-objective optimization (NSGA-II and SPEA2) with statistical confidence (see the "+" symbols in the last column of the table).

If we now turn to analyze the distribution of the computed solutions along the Pareto front (Table 9.7) it is still more clear than before that aMOCell4 is outperforming them in all the tested problems. For the ZDT benchmarking functions aMOCell4 is for this metric even one order of magnitude better than the other techniques. The reader can find in Fig. 9.1 the best found Pareto front (out of 100 independent runs) for WFG6; with it we want to illustrate how good is the cellular based approach compared against the rest of algorithms in computing diverse points evenly spread along the front, a desired feature in present proposals for MO algorithms.

Regarding the metric HV the reader can find the numerical performance in Table 9.8. As in the previous cases, aMOCell4 gets the best (largest) values in 10 (out of 14) problems. It is again the best technique for the ZDT family and outperforms the rest for 5 (out of 9) of the WFG functions. We again remind that the small differences are statistically significant: they are small because of the normalization process but clearly important, as it can be noticed in Fig. 9.1.

Let us now take our attention to the whole set of results (the recently mentioned tables). The global analysis of them throws the clear conclusion that the cellular based proposal, aMOCell4, performs better than NSGA-II and SPEA2. This is an outstanding result, since these two techniques are being used by a huge number of researchers that now could profit from our new proposal to solve their problems. It is clear that the underlying cellular-like search is a valuable component for solving complex problems also in the multi-objective domain.

9.4 Conclusions

We started this chapter by proposing a simple extension of the basic canonical cGA for multi-objective problems: MOCell. MOCell is by itself an extension of a former proposal called cMOGA [8, 18], the early proposal we did in this field. The key structural issue in MOCell is the utilization of an external archive to store the non-dominated individuals found during the search process, and more precisely its utilization as a source for solutions (feedback from archive to algorithm) during the evolution towards an optimal Pareto front. The feedback process has shown in all our past experience to be a milestone for success.

In the chapter we have later analyzed six extensions of MOCell in order to fine tune it and to show how far the cellular concepts can help the researcher to define very efficient and accurate algorithms. The three concepts that showed the larger benefits are the asynchronous update of individuals (line sweep), the replacement of the worst individual of the considered neighborhood, and finally the idea of using the external archive as the source for selecting one of the parents in every iteration. The result is aMOCell4, a multi-objective optimization engine able of unseen performances in a large testbed and in competition with the best, state-of-the-art, algorithms NSGA-II and SPEA2.

This chapter indicates just a line of research in combining cellular GAs and multi-objective concepts. There are other many possible ideas to be explored in the future, maybe involving local search, hybridization, parallelization, and other open issues that could help in creating powerful algorithms to solve problems of larger complexity and larger dimension than the ones solved at present.

10

Other Cellular Models

Imagination is more important than knowledge.

Albert Einstein (1879 - 1955) – Physicist

After proposing so many cellular models in the previous chapters, we now proceed in this chapter to complete this extensive study by compiling some other interesting cellular models recently proposed in the literature.

Specifically, we consider two new families of cellular EAs. On the one hand, it is presented in Sect. 10.1 a new family of cellular GAs in which a hierarchy is established inside the population such that better individuals are moved to the same region of the grid. This way, a faster convergence of the best individuals is promoted, while the diversity is maintained in locations far away from this region. This new hierarchical cellular model is applied here to GAs, but it can be implemented in any other family of EA. It can be helpful in parallel implementations looking for fail tolerance, since some parts of the population could be lost with negligible effects on the search. Also, faster convergence can be an interesting outcome of this new mode.

On the other hand, a new class of estimation of distribution algorithms (EDAs) is addressed in Sect. 10.2. This new family of algorithms is known as cellular EDAs (cEDAs), and they are characterized by their decentralized population, which is partitioned into many small collaborating sub-populations, arranged in a toroidal grid, and interacting only with its neighboring sub-populations.

These two new families of cellular models are analyzed and compared in this chapter against their equivalent panmictic algorithms from the points of view of both theory and practice.

10.1 Hierarchical cGAs

In this section we describe a new kind of cGA, called hierarchical cGA (H-cGA) [139], where the population structure is augmented with a hierarchy according to the current fitness of the individuals. Better individuals are moved towards the center of the grid, so that high quality solutions are exploited quickly, while at the same time new solutions are provided by individuals

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_10, © Springer Science+Business Media, LLC 2008

Algorithm 10.1 Pseudo-code of H-cGA

```
1. proc Evolve(hcga)        //Algorithm parameters in 'hcga'
2. GenerateInitialPopulation(hcga.pop);
3. Evaluation(hcga.pop);
4. while !StopCondition() do
5.    for individual ← 1 to hcga.popSize do
6.       neighbors←GetNeighbors(hcga,position(individual));
7.       parents←Select(neighbors);
8.       offspring←Recombination(hcga.Pc,parent1,parent2);
9.       offspring←Mutation(hcga.Pm, offspring);
10.      Evaluation(offspring);
11.      Replacement(position(individual),hcga,auxiliary_pop,offspring);
12.   end for
13.   hcga.pop←auxiliary_pop;
14.   Swap_Operation(hcga.pop);    // Keep the hierarchy of the population
15. end while
16. end proc Evolve;
```

at the outside that keep exploring the search space. This algorithmic variant is expected to increase the convergence speed of the cGA algorithm and maintain the diversity given by the distributed layout. We here examine the effect of the introduced hierarchy by observing the variable takeover rates at different hierarchy levels. Additionally, we compare the H-cGA to the cGA algorithm on a set of benchmark problems, and show that the new approach obtains a promising performance.

In Alg. 10.1 we show a pseudo-code of a canonical hierarchical cGA. As it can be seen, it is similar to a canonical cGA except that in the case of the H-cGA algorithm the population is re-arranged after every generation with the hierarchical swap operation (line 15). In Sect. 10.1.1 it is addressed the hierarchical ordering of the population that is introduced for the H-cGA algorithm and how this ordering is obtained. After that we introduce in Sect. 10.1.2 a new selection operator for this algorithm. This new algorithm is evaluated both theoretically and in practice in Sects. 10.1.3 and 10.1.4, respectively.

10.1.1 Hierarchy

The hierarchy is imposed on the cellular population of the GA by defining a center at position $(x/2, y/2)$ and assigning hierarchy levels according to the distance from the center. The center has level 0 and the level increases with increasing distance to the center (as it is displayed in Fig. 10.1). The hierarchy is updated after each iteration of the cGA and individuals with high fitness are moved towards the center. Note that the population topology is still toroidal when selecting parents.

In Fig. 10.1 we show how this swap operation is performed. It is applied between cells indicated by the arrows, in the order denoted by the numbers

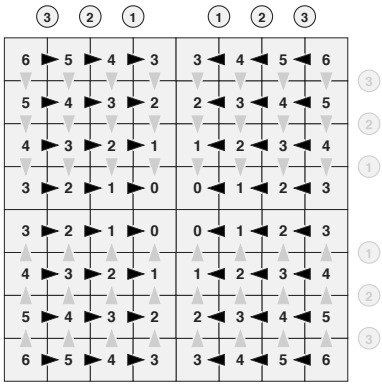


Fig. 10.1. The H-cGA and its different hierarchy levels

outside of the grid. The update of the hierarchy is performed alternatingly horizontally (black) and vertically (grey) by the swap operation. We assume an even number for the population dimensions x and y, so that the population can be uniquely divided into left (upper) and right (lower) half, for the horizontal (vertical) swap. This implies that there are 4 individuals in the center of the population, i.e., on the lowest level of the hierarchy (that with the best individuals).

In the following we describe the horizontal swap operation, the vertical swap is done accordingly. Each individual (i, j) in the left half compares itself with its left neighbor $(i-1, j)$ and if this one is better they swap their positions. These comparisons are performed starting from the center of the grid towards the outside, see Fig. 10.1. Thus, at first individuals in columns $\frac{x}{2} - 1$ and $\frac{x}{2} - 2$, for $i = 0, \ldots, y$, are compared. If the fitness value at position $(i, \frac{x}{2} - 2)$ is better they swap positions. These pairwise comparisons are then continued towards the outside of the grid. Hence, an individual can improve only one level at a time but can increase several levels within one iteration.

10.1.2 Dissimilarity Selection

The proposed hierarchy promotes the recombination of good individuals within the population. In this respect, H-cGA is similar to a panmictic GA with a fitness-biased selection. In our H-cGA algorithm this selective recombination of the elite individuals of the population is already included in the hierarchy. Therefore, we examined a new selection operator that is not based on the relative fitness of the neighbouring individuals but instead considers the difference between the respective solution strings. As for the binary tournament (BT) selection, two neighbors are selected randomly, but in contrast to BT, where the better one is selected, the one that is more different from the current individual is selected. All the considered problems are binary encoded, hence we use the Hamming distance for determining the dissimilarity.

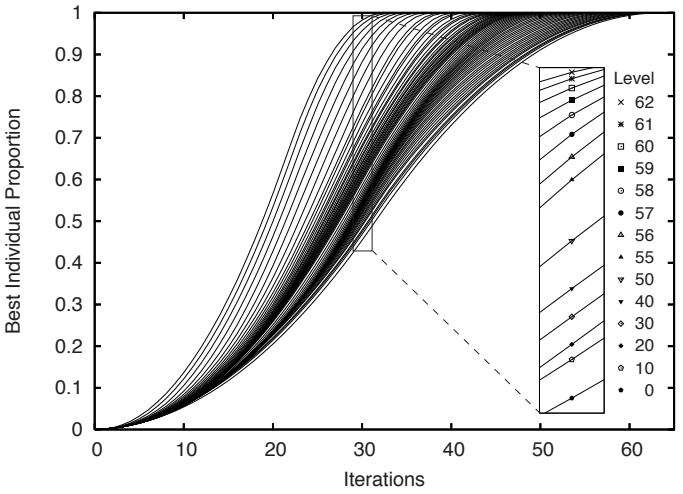


Fig. 10.2. Takeover curve resulting from initially placing the best individual at different levels and evolving the population only with a selection operator

The overall optimization progress of the algorithm is ensured by only replacing an individual if the newly generated individual, by crossover and mutation, is better than the previous one.

10.1.3 First Theoretical Results: Takeover Times

We are providing in this section a closer examination of the properties of the proposed algorithm by studying its *takeover* time (see Chap. 4) and comparing it to that of a canonical cGA. First, we are looking at a deterministic takeover process, where the best individual within the neighbourhood is always selected. Later, we also consider BT selection and the newly proposed dissimilarity selection. Initially all individuals get assigned random fitness values from [0:4094] and one individual gets the maximum fitness value of 4095. Then the selection-only algorithm is executed and the proportion of the entire population that holds the maximum fitness value at each iteration is recorded. The considered grid is of size 64×64 and hence the population consists of 4096 individuals.

In the H-cGA the different levels of the hierarchy influence the time required for takeover. In order to accurately determine this influence, we use the deterministic selection operator. The best individual is initially placed on each possible position on the grid and the takeover time for these 4096 different setups is measured. Then the results for all positions on a specific level of the hierarchy are averaged to obtain the takeover rate for introducing the best individual at this particular level. In Fig. 10.2 the obtained takeover

Table 10.1. Takeover times for the algorithms with BT and dissimilarity selection

Algorithm	Avg. Iterations	Min – Max
cGA	75.2 ± 1.5	72.0 – 80.0
cGA-Dis	78.7 ± 1.7	75.0 – 83.0
H-cGA	71.0 ± 6.3	48.0 – 81.0
H-cGA-Dis	79.6 ± 3.8	71.0 – 89.0
H-cGA level 0	81.5 ± 1.5	78.0 – 87.0
H-cGA level 62	46.3 ± 2.1	42.0 – 57.0

curves are shown for introducing the best individual at different levels of the hierarchy. The slowest takeover rate is achieved when placing the best value at the center of the grid on level 0. This takeover rate is identical to the deterministic takeover for the regular cGA. The hierarchy level 62 consists of the 4 cells on the corners of the grid, where the fastest takeover rate is obtained. This increasing takeover speed with increasing hierarchy level is very regular, as can be seen in the detail display of iteration 30. The reason why having the best individual near the outside of the hierarchy accelerates takeover is, that, since for selection the topology is still toroidal, adjacent individuals on the opposite end of the grid also adopt the highest fitness value at the beginning of the run. Then the hierarchy swap operation moves the maximal value towards the center from several sides and therefore the actual takeover speed is increased.

We also measured the time required for takeover with BT and dissimilarity selections. The best individual was placed at a random position and the experiments were repeated 100 times. For the experiments with dissimilarity selection, the individuals are using a binary string that corresponds to the 12 bit representation of their current fitness value. Our results are shown in Table 10.1. Specifically, we provide the average number of iterations, the standard deviation, and the minimum and maximum values obtained. As it can be seen, initially placing the best individual at the center (H-cGA level 0) slows down the takeover compared to the regular cGA algorithm. This is because once the maximal value spreads it will be delayed until all of the central cells hold the best fitness value, otherwise it will be swapped towards the center again. In the two algorithms, the use of the BT selection induces a higher selection pressure than in the case of using the dissimilarity selection. After applying some statistical tests (see Chap. 5) to the results in Table 10.1, we obtained that there exist statistically significant differences at a 95% confidence level.

10.1.4 Computational Experiments

We complete here the study made in Sect. 10.1.3 by analyzing the behavior of the proposed H-cGA and its equivalent cGA on a selected benchmark of problems. As it is usual in the studies made in this book, this benchmark is com-

Table 10.2. Parameterization used in our algorithms

Population	20×20 Individuals
Parent selection	Current individual + (BT or dissimilarity)
Recombination	DPX, $p_c = 1.0$
Bit mutation	Bit-flip, $p_m = 1/L$ (L = Individual length)
Replacement	Rep_if_Better
Stopping criterion	Find optimum or reach 2500 generations

posed of problems with many different features, such as multimodality, deceptiveness, use of constraints, or problem generators. They are Onemax (with size 500), the Massively Multimodal Deceptive Problem (MMDP), P-PEAKS, and the Minimum Tardy Task Problem –MTTP– (instances of 20, 100, and 200 tasks). All the details on these problems are given in Appendix A.

The same parametrization is used for the two algorithms (see Table 10.2), and both BT and dissimilarity selection have been tested. We used a population of 400 individuals arranged in a grid of size 20×20 with a NEWS neighborhood (the cell itself and its North, East, South and West neighbours are considered). In all our experiments, one parent is the center individual itself and the other parent is selected either by BT or dissimilarity selection (it is ensured that the two parents are different). An individual is replaced only if the newly generated fitness value is better. The recombination method used is the two point crossover (DPX), and the selected offspring is the one having the largest part of the best parent. The crossover probability is 1.0 and bit mutation is performed with probability $1/\#bits$ for genome string of length $\#bits$. In order to have statistical confidence, all the presented results are average over 100 runs, and the analysis of variance –ANOVA– statistical test (or Kruskal-Wallis if the data is not normally distributed) is applied to the results. A 95% confidence level is considered.

In Table 10.3 we present the results we have obtained for all the test problems. Specifically, we show the success rate (number of runs in which the optimum was found), and some measures on the number of evaluations made to find the optimum, such as the average value, the standard deviation, and the maximum and minimum values. The results of our statistical tests are in column *Test*, where symbol '+' means that there exist statistically significant differences. The evaluated algorithms are cGA and H-cGA both with BT and dissimilarity selections.

As can be seen in Table 10.3, both the cGA and the H-cGA algorithms were able to find the optimal value in each run for all the problems, with the exception of MMDP. For this problem, the cGA algorithm achieves slightly better success rates than H-cGA. Regarding the average number of evaluations required to reach the optimum, the hierarchical algorithm always outperforms cGA, except for the P-PEAKS problem. Hence, the use of a hierarchical population allows us to accelerate the convergence speed of the algorithm to the optimum, while it retains the interesting diversity management of the canonical cGA.

Table 10.3. H-cGA versus its equivalent cGA (BT and dissimilarity selections)

Problem	Algorithm	Success	Avg. Iterations	Min–Max	Test
Onemax	cGA	100%	$129.4_{\pm 7.3}$	111.2–145.2	+
	cGA-Dis	100%	$140.7_{\pm 8.1}$	121.6–161.2	
	H-cGA	100%	$94.1_{\pm 5.0}$	83.2–106.4	
	H-cGA-Dis	100%	$103.1_{\pm 5.6}$	90.4–116.8	
MMDP	cGA	67%	$202.4_{\pm 154.7}$	120.8–859.2	+
	cGA-Dis	97%	$179.8_{\pm 106.3}$	116.8–846.0	
	H-cGA	55%	$102.6_{\pm 76.1}$	68.8–652.8	
	H-cGA-Dis	92%	$122.3_{\pm 111.7}$	73.2–837.6	
P-PEAKS	cGA	100%	$41.9_{\pm 3.0}$	32.0–48.4	+
	cGA-Dis	100%	$52.9_{\pm 5.2}$	38.4–66.0	
	H-cGA	100%	$47.2_{\pm 8.6}$	30.8–71.2	
	H-cGA-Dis	100%	$81.1_{\pm 17.1}$	45.2–130.8	
MTTP-20	cGA	100%	$5.1_{\pm 1.2}$	1.6–8.0	+
	cGA-Dis	100%	$6.0_{\pm 1.3}$	2.0–9.2	
	H-cGA	100%	$4.7_{\pm 1.1}$	1.6–7.2	
	H-cGA-Dis	100%	$5.5_{\pm 1.2}$	2.8–8.0	
MTTP-100	cGA	100%	$162.2_{\pm 29.3}$	101.6–241.6	+
	cGA-Dis	100%	$174.6_{\pm 26.3}$	96.4–238.8	
	H-cGA	100%	$138.3_{\pm 35.4}$	62.0–245.6	
	H-cGA-Dis	100%	$132.4_{\pm 26.2}$	64.0–186.8	
MTTP-200	cGA	100%	$483.1_{\pm 55.3}$	341.6–632.4	+
	cGA-Dis	100%	$481.0_{\pm 71.6}$	258.8–634.8	
	H-cGA	100%	$436.2_{\pm 79.7}$	270.4–631.2	
	H-cGA-Dis	100%	$395.3_{\pm 72.6}$	257.6–578.8	

If we now compare the results of the algorithms when using the two different studied selection schemes, we notice that with the dissimilarity selection the success rate for the MMDP problem can be increased for both the cGA and the H-cGA algorithm. In terms of efficiency, the algorithms are usually worse when using the dissimilarity selection.

In Fig. 10.3 we plot the expected number of evaluations, defined as the average number of evaluations divided by the success rate, required to find the optimal value for each problem. The displayed results are relative to the expected number of evaluations for the cGA.

As we already concluded analyzing the results in Table 10.3, it can be easily seen in Fig. 10.3 that the expected number of evaluations is increased when using the dissimilarity selection compared to the equivalent algorithms with BT. For the cGA the dissimilarity selection was able to reduce the number of required evaluations only for the MMDP problem. But for the H-cGA it proved to be useful also for the two largest MTTP instances. In general, the expected number of evaluations is lower for the two studied versions of H-cGA.

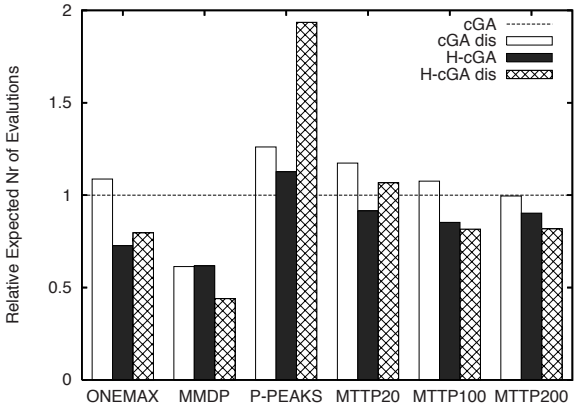


Fig. 10.3. Expected number of evaluations required to reach the optimum, relative to the steps required by cGA, for the different benchmark problems

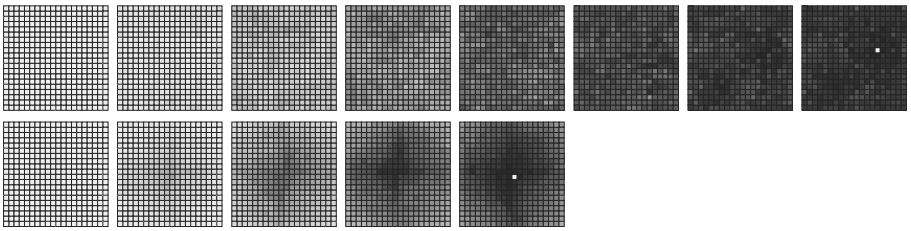


Fig. 10.4. Evolution of the population for the cGA (top) and the H-cGA (bottom)

Finally, in order to illustrate the effects of using a hierarchical population in the cGA, we show a sample run of the cGA (top) and the H-cGA (bottom) algorithms in Fig. 10.4. The pictures are snapshots of the population taken every 50 iterations for the MDDP problem until the optimum is found (iteration 383 for cGA and 233 for H-cGA). The darker an individual is colored the higher its fitness is; the white cell in the last image contains the optimal solution to the problem (maximum fitness). As it can be seen, the H-cGA algorithm quickly focuses on promising solutions, while at the same time different solutions of lower quality are kept at the outside of the hierarchy.

10.2 Cellular Estimation of Distribution Algorithms

In this section we present and analyze a new class of estimation of distribution algorithms (EDAs) [158, 187], called cellular EDAs [174]. EDAs are an alternative family to traditional EAs in which a different kind of variation operators is used. The successive generations of individuals are created by using estimations of distributions observed in the current population instead of evolving

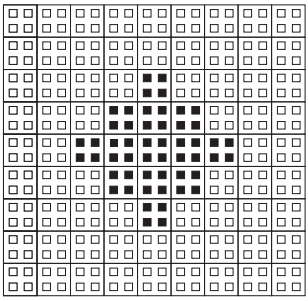


Fig. 10.5. A cEDA with a C13 neighborhood and the population shape $2 \times 2 - 9 \times 9$

the population with the typical variation operators (like crossover and mutation) used in other EAs. Hence, the main feature distinguishing EDAs from other more classical EAs is that EDAs learn the interactions among variables (building blocks) in the problem to be solved. At the same time, it is the main drawback of EDAs due to the complexity of this learning and simulation task.

Cellular EDAs were introduced as a decentralized version of EDAs, and also as a generalization of the cellular models developed for other evolutionary algorithms [35, 176]. In a cEDA the population is decentralized by partitioning it into many small collaborating sub-populations (called *cells* or *member algorithms*), arranged in a toroidal grid, and interacting only with the neighboring sub-populations. One distinctive feature of this class of algorithm is that selection is decentralized at the level of the member algorithms, while in other cellular EAs it usually occurs at the recombination level.

The organization of cEDAs is based on the traditional 2D structure of overlapped neighborhoods. That structure is better understood in terms of two grids, i.e., one consisting of strings and another consisting of disjoint sets of strings (cells). Fig. 10.5 shows a global population of 18×18 strings (small squares) partitioned in a 9×9 toroidal grid of cells (large squares) containing 4 strings each. The neighborhood used is the so called C13, which is composed by the considered subpopulation plus its 12 nearest cells (measured with the manhattan distance). We adopt the same notation used in [174] for describing the shape of the population: it consists of the shape of cells in terms of strings plus the shape of the whole population taking into account the cells (composed of one or more strings). For example, following this nomenclature, the grid of Fig. 10.5 is labelled as $2 \times 2 - 9 \times 9$.

In Alg. 10.2 we present a pseudo-code of the proposed cEDA approach. Each iteration of a cEDA consists of exactly one iteration of all the member algorithms. Each of these member algorithms is responsible for updating exactly one subpopulation, and this is made by applying a local EDA model to the population composed of its strings and those of its neighbor sub-populations (lines 7 to 9). The implementation of cEDA carried out in this paper is *synchronous*, since the successive populations replace each other

Algorithm 10.2 Pseudo-code of a simple cEDA

```
1. proc Evolve(ceda)        //Algorithm parameters in 'ceda'
2. Set t ← 1;
3. GenerateInitialPopulation(ceda.pop);
4. Evaluation(ceda.pop);
5. while !StopCondition() do
6.    for every cell do
7.       Select locally M ≤ SizeOf(Neighborhood) × SizeOf(cell) strings of the
         neighborhood according to a selection method;
8.       Estimate the distribution p^s(x,t) of these M selected strings;
9.       Generate SizeOf(cell) new points according to the distribution p^s(x,t);
10.      Insert the generated points in the same cell of an auxiliary population;
11.   end for
12.   Replace the current population with the auxiliary one;
13.   Compute and update the statistics;
14.   Set t ← t + 1;
15. end while
16. end proc Evolve;
```

at once (line 12), so the new individuals generated by the local learning and sampling steps are placed in a temporal population (line 10).

In the replacement step (line 12), the old population can be taken into account (i.e., replacing a string if the new one is better) or not (always adding the new string to the next population). The first issue (called *elitism*) is the preferred one for this study. Finally, computing basic statistics (line 13) is rarely found in the pseudo-codes of other authors in the EA field. However, it can be used for monitoring the algorithm and decide changes in the adaptive search, when needed.

A critical issue in a cEDA is the computation of the probabilistic model due to the high computational cost it usually supposes. The reader is referred to [174] for an explanation of several alternative learning schemes for cellular EDAs.

In next section we compare the Univariate marginal distribution algorithm (UMDA) versus its cellular version (cUMDA, see [23]) both theoretically and in practice. The UMDA was presented for the first time by Müehlenbein and Paaß in [187], and it is one of the simplest algorithm in the EDAs family. In UMDA, it is considered that variables are independent from the others, so there are no dependencies between them. The current generation evolves towards the new one by computing the frequencies of values of the variables on each position in the selected set of promising solutions. These frequencies are then used to compute new solutions, which replace the old ones. Due to its simplicity, it is a very efficient algorithm (converges quickly), and its behavior is particularly good for linear problems. Notice that the use of UMDA as the member algorithm, implies that we do not need to learn the structure of the model (it is known), but just the univariate marginal frequencies.

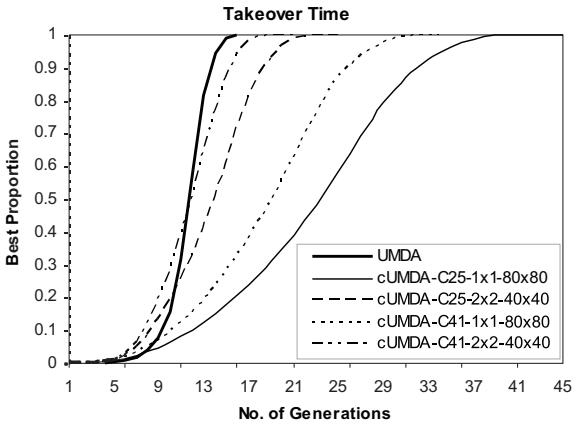


Fig. 10.6. Takeover curves for the studied algorithms

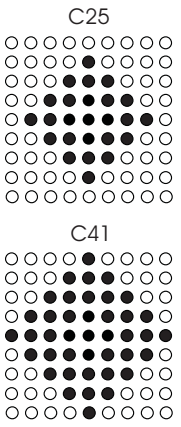


Fig. 10.7. Neighborhoods

10.2.1 First Theoretical Results: Takeover Times

We include in this section a brief study of the theoretical behavior of cUMDAs in terms of their selection pressure. This is the first time (at the very best of our knowledge) that the theoretical behavior of EDAs has been studied using the takeover time. For that, we compute the proportion of the best individuals in the population (after truncating), and use it for generating the new one with elitism (best individuals are kept).

In Fig. 10.6 we plot the growth curves of UMDA and four different cUMDAs. In order to obtain meaningful results, we use a large population of up to 64000 individuals for this study. The cUMDAs differ both in the neighborhood used and in the population structure. Thus, we used a population structured in 80×80 cells of 1 individual each, or 40×40 cells composed by 4 individuals. Neighborhoods C25 and C41 (see Fig. 10.7) are used with the two different population structures, and they are composed by the considered cell plus the nearest 24 (for C25) and 40 (in the case of C41) cells measured in Manhattan distance. As it can be seen, the highest selection pressure (shorter takeover time) corresponds to UMDA. Regarding the cUMDAs, the two algorithms having one individual per cell are those with lower selection pressure (longer takeover time). The reason is that, in the cases of the cUMDAs with four individuals per cell, the exploration capabilities of the algorithm are penalized since the number of individuals in the neighborhood is multiplied by four.

10.2.2 Computational Experiments

In this section we turn to present and analyze the results we have obtained in our experiments for a benchmark composed by five problems with different features, namely Onemax (size 1000), Plateau, IsoPeak, P-PEAKS, and MTTP problems. All of them are described in Appendix A. As in the previous section,

Table 10.4. Parameterization used in all the compared algorithms

Population size	400 individuals
Parent selection	Truncation selection, $\tau = 0.5$
Mutation	Bit-flip, $p_m = 1/L$ (L = Individual length)
Replacement	Replace-if-Better
Stop condition	Optimum reached or 400,000 fitness evaluations
Member algorithm	UMDA
Cellular case	*Neighborhood shape:* $C25$ and $C41$
	Population shape: $1 \times 1 - 20 \times 20$ and $2 \times 2 - 10 \times 10$

the algorithms we have studied in our comparison are the standard UMDA plus four different proposed cellular versions. Their parameterization is given in Table 10.4. The studied cUMDAs are those using the C25 neighborhood and one or four strings per cell (called C25-1×1-20×20, and C25-2×2-10×10, respectively), and these same two population structures with the neighborhood C41: C41-1×1-20×20 and C41-2×2-10×10. All the algorithms have populations of 400 individuals. The selection method used is the truncation selection (with $\tau = 0.5$), applied into the local subpopulation pool. Finally, all the algorithms implement a mutation step in order to introduce some diversity into the population. Although it is not usual in EDAs, we apply this mutation due to the good results obtained in some preliminary experiments. The proposed mutation simply consists in flipping the genes of the newly generated individuals with probability $1/L$ (being L the length of the chromosome). The termination criterion is either to find an optimal solution or to make 400,000 fitness function evaluations.

In order to get statistically significant results, we have made 100 runs for each test, and computed the analysis of variance (ANOVA) or Kruskal-Wallis tests for comparing our results (depending on whether the data follows a normal distribution or not). For these statistical tests, we consider in this work a significance level of 95% (p-value under 0.05). Both UMDA and cUMDA are written in C++ and executed in a Pentium 4 2.4 GHz with Linux having 512 MB of RAM.

We present in Table 10.5 the percentage of runs in which the algorithms found the optimal solution to the problems (success –or hit– rate). As it can be seen, UMDA has difficulties for finding the optimum in the case of MTTP (74% of the runs), and was not able to get it in any run for the IsoPeak problem. Conversely, its four cellular versions studied here did not find any difficulty for obtaining the optimal value for all the problems in every run (100% of success rate).

Table 10.5. Success rate of UMDA and the four compared cUMDAs

Algorithm	**OneMax**	**Plateau**	**IsoPeak**	**P-PEAKS**	**MTTP**
UMDA	100%	100%	0%	100%	74%
cUMDA C25-1×1-20×20	100%	100%	100%	100%	100%
cUMDA C25-2×2-10×10	100%	100%	100%	100%	100%
cUMDA C41-1×1-20×20	100%	100%	100%	100%	100%
cUMDA C41-2×2-10×10	100%	100%	100%	100%	100%

Table 10.6. Function evaluations of UMDA and the four compared cUMDAs

Neighborhood	**OneMax**	**Plateau**	**IsoPeak**	**P-PEAKS**	**MTTP**
UMDA	26072.0	18972.0	—	15040.0	169713.5
	±463.2	±1054.9	—	±1448.4	±106624.5
cUMDA C25-1×1-20×20	23298.1	16337.8	176878.7	**13461.0**	14976.4
	±382.8	±584.9	±36384.0	±1650.0	±1348.0
cUMDA C25-2×2-10×10	22037.6	14961.2	218190.0	34994.2	14418.1
	±346.7	±511.7	±47518.1	±13342.2	±1116.7
cUMDA C41-1×1-20×20	22500.9	15454.9	**176138.8**	16915.1	14469.9
	±361.1	±510.1	±41834.8	±3091.6	±981.4
cUMDA C41-2×2-10×10	**21851.7**	**14773.6**	253725.4	41795.3	**14235.5**
	±340.3	±469.2	±58172.6	±15362.1	±1203.6
Test	+	+	+	+	+

In Table 10.6 we present the average number of evaluations and the standard deviation needed by the five studied algorithms for solving the problems. In the last row of the table we present the p-values obtained in our statistical tests when comparing all the algorithms for each problem. The '+' symbol stands for statistical confidence in the comparison of the five algorithms, i.e., the results of almost two of the compared algorithms are statistically different. As an important result, the most efficient algorithm (lowest number of evaluations) for every problem (**bolded** values) is always one of the studied cellular versions of UMDA.

In Fig. 10.8 we graphically show the results of Table 10.6. It is easy to see that the less efficient algorithm for all the problems is UMDA, with the exception of P-PEAKS. With respect to the different cellular versions of UMDA, cUMDA C41-2×2-10×10 is the most efficient for three out of the five studied problems (OneMax, Plateau, and MTTP), although it is the worst one for the two other problems. However, the differences among the studied cUMDAs are, in general, very low (no statistical confidence was found in the comparison of the cUMDAs).

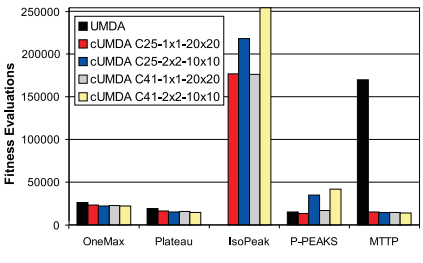


Fig. 10.8. Efficiency of the algorithms

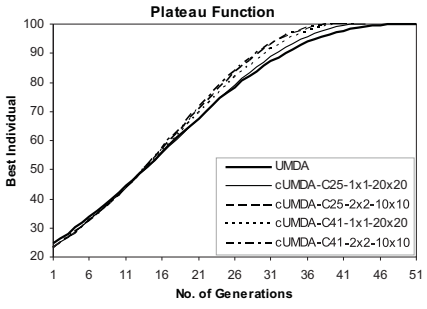


Fig. 10.9. Evolution of the best fitness value for Plateau

Finally, in Fig. 10.9 we plot an example of the evolution of the best fitness value for UMDA and the 4 proposed cUMDAs when solving the Plateau problem. The value plotted in each generation is computed as the average of 100 executions. As can be seen, the two algorithms that converge earlier to the optimum are the two cUMDAs with the population 2×2-10×10, which show a similar behavior (almost indistinguishable), as we previously obtained in the analysis of the takeover time. The slowest of the studied algorithms for this problem is UMDA, which finds more difficulties than the cUMDAs for avoiding local optima and converge to the global optimum due to its higher selection pressure.

10.3 Conclusions

As a complement to the previous chapters, we present in this chapter two more new families of cellular evolutionary algorithms. The first one is the so called hierarchical cGA, or H-cGA. In H-cGAs, we included the idea of establishing a hierarchy among the individuals of the population of a canonical cGA. With this hierarchical model we achieve different levels of the exploration/exploitation tradeoff of the algorithm in distinct zones of the population simultaneously. We studied these specific behaviors at different hierarchy levels by examining the respective takeover rates. Additionally, we have compared the H-cGA with two different selection methods to the equivalent cGAs and the hierarchical algorithm performed better on almost all test functions. The newly proposed dissimilarity selection was not useful in all the scenarios since it promotes diversity into the population but, as a consequence, the convergence is usually slower.

In the second part of this chapter we have investigated an algorithm from a new class of decentralized EDAs, called cellular EDA, based on the functioning of other existing cellular EAs. Four approaches based on UMDA, a simple EDA, have been tested. As a result, the comparison between the four new cUMDAs and UMDA reports very advantageous results for the cellular models, since UMDA (centralized) is, in general, the worst algorithm both in terms of efficacy (success rate) and efficiency (number of evaluations to reach an optimum) for all the problems.

11

Software for cGAs: The JCell Framework

Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger Dijkstra (1930 - 2002) – Mathematician

A new object oriented framework, called JCell, has been built along with the works carried out for this book. This framework implements most of the new algorithmic models addressed in this book, and also the problems studied. Our purpose for the development of JCell is twofold. Firstly, we provide a tool for easily reproducing the reported results in this book with a really low effort. Secondly, we intend to provide the scientific community with a large object oriented library for working with cEAs and the most recent advances in this field. For this objective, we consider it is mandatory to make an understandable and easy to use code, and thus much attention has been paid to the design of JCell.

Currently, only GAs have been implemented in JCell, but it is very easy and intuitive to extend it with some other kind of EA, as we will see along this chapter. Thus it is really easy to migrate all the algorithmic improvements proposed in this book to other fields of evolutionary computation with a low effort. Additionally, several population structures are implemented in JCell, so it is possible to configure both traditional panmictic (generational and steady state) and decentralized (cellular and islands) EAs. This allows to easily make studies of the advantages of using the structured cellular populations versus other models.

JCell is freely available at `http://neo.lcc.uma.es/Software/JCell`.

This chapter has the following structure; in Sect. 11.1 we describe the design of JCell, while in Sect. 11.2 a brief handbook of the using of the framework is presented. Finally, we conclude in Sect. 11.3.

11.1 The JCell Framework

We present in this section JCell. As it is said hereinbefore, JCell is a generic framework for working (mainly) with cGAs, containing the last advances in the field of cellular algorithms. In addition to the cGA, JCell implements two panmictic algorithms, as the generational and the steady state GAs, and a

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_11, © Springer Science+Business Media, LLC 2008

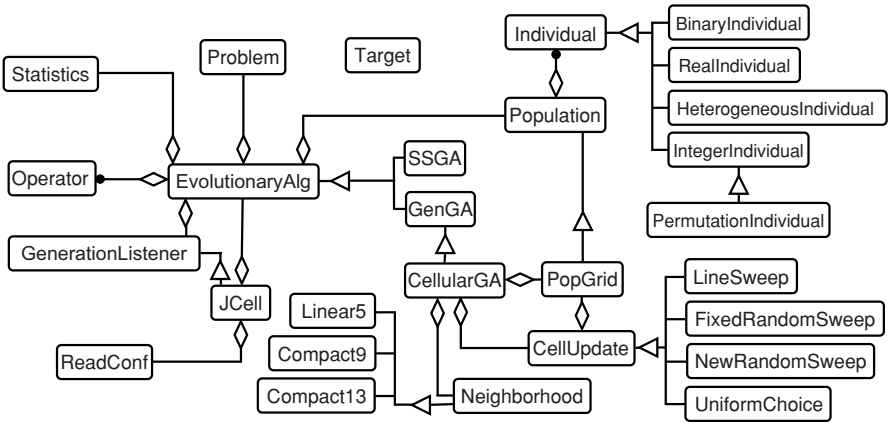


Fig. 11.1. Brief description of JCell object oriented design

sequential version of a GA having the population structured in islands. The use of JCell is very easy, since it is only required the manipulation of a simple configuration file. JCell allows the user to work in combinatorial optimization problems, integer programming, continuous optimization, or even multi-objective environments, all this with or without constraints. Moreover, most of the new algorithmic models addressed in this book are also implemented in JCell.

We consider JCell a really interesting and useful tool for future research, since it allows the combination of all the new promising techniques developed in this book. Additionally, its careful design following the software engineering recommendations provides an intuitive code, allowing the user to easily make modifications and/or add new features to the framework.

JCell is implemented in Java [1, 82]. We selected an object oriented language because it provides an easier and simpler design of our framework, what facilitates to be used by other researchers. Moreover, if we use an object oriented language, we can benefit from characteristics as the easy code reutilization, the inheritance, the overload or the polymorphism. All these features allow us to develop a clear code with a simple design. Among the object oriented languages we chose Java basically because of its portability (which is a very important feature for the high diffusion JCell is intended to have), and also to the large variety of frameworks available, what makes the programming task easier. The portability of Java is possible because it is a semi-interpreted language, and thus for running a Java program in any kind of computer it is only required to have the *Java Virtual Machine* –the Java interpreter– installed. JCell is compatible from version JDK 1.5 ahead.

In Fig. 11.1 we show a summarized UML diagram describing the design of JCell. The class JCell is the main class of our frame, as it contains the main() method for executing the algorithm. JCell implements the GenerationListener

interface, which has a single declaration of function (called generation()). This interface is used for communicating the class which performs the reproductive cycle of the GA (EvolutionaryAlg) and JCell: after each generation, EvolutionaryAlg calls the function generation() of JCell. In this way, the programmer is able to obtain any information of the algorithm in each generation, or even to perform any change during the execution for obtaining easily variations on the implemented GAs.

JCell contains ReadConf, which is in charge of reading all the required parameterization from the configuration file, and set the corresponding values to the parameters of EvolutionaryAlg. Thus, the first thing we have to do when using this framework in our program must be a call to ReadConf with the path to a configuration file wherein the desired parameterization of the algorithm is given. The structure of this configuration file is described in Sect. 11.2.

One of the most important classes of JCell is EvolutionaryAlg. As it can be seen, it is a generic class for the design of evolutionary algorithms. In this class all the information the algorithm needs for executing is stored, as the population, the genetic operators and their probabilities of application, the synchronous or asynchronous individuals update, the number of generations (and evaluations) performed, the termination condition of the algorithm, the algorithmic model to use (multi-objective, hierarchical, or adaptive are the ones currently implemented), etc. In the following paragraphs we describe the classes referenced by EvolutionaryAlg: Population, Neighborhood, Problem, Operator, and Statistics, besides GenerationListener, which was previously commented.

Population contains a linear list of Individual objects, besides the methods which implements the operations the EAs need to do on the population, as obtain or insert an individual, create a new population, etc. Individual is the class of JCell implementing a generic individual of an EA, providing the main operations usually performed on individuals (obtain and set the fitness value, the values of its genes, etc.). The genes of the chromosome of Individual are of Object type, and depending on the genotype information hold by the individual ($G = \{g_1, g_2, \ldots, g_l\}$, being l the length of the chromosome), it can be instantiated as:

- BinaryIndividual. Class which inherits from Individual where the chromosome is composed by logic values (in Java, objects of Boolean type). This type of individuals is used in combinatorial optimization problems.
- RealIndividual. This class was created for the case of numeric optimization. The genes are real numbers (type Double of Java), and inherits from Individual.
- IntegerIndividual. As it happens in the two previous classes, IntegerIndividual inherits from class Individual. In this case, the genes are integer numbers (class Integer).

Table 11.1. Individuals implemented in JCell

Class	Genotype	Inherits from
BinaryIndividual	$G = \mathbb{B}^l$	Individual
RealIndividual	$G = \mathbb{R}^l$	Individual
IntegerIndividual	$G = \mathbb{N}^l$	Individual
HeterogeneousIndividual	$G = (\mathbb{B} \cup \mathbb{R} \cup \mathbb{N})^l$	Individual
PermutationIndividual	$G = \mathbb{N}^l; g_i \neq g_j,\ 0 \leq i, j \leq l$	IntegerIndividual

- HeterogeneousIndividual. This class inherits from Individual as the previous ones, and in this kind of individual it is allowed to have in its chromosome genes of different types.
- PermutationIndividual. It is an IntegerIndividual where the values of the genes are a permutation of integer numbers, so two genes can not have the same value in a chromosome.

A summary of the different kinds of individuals available in JCell is given in Table 11.1. Particularly, it is given, for each type of individual available, the class it belongs to, the composition of the genotype, and the class it inherits from. The fitness value of an individual is an array of *Double* objects. This array has length 1 when we are solving problems with only one function to optimize, and n in the case of multi-objective optimization (n being the number of objectives).

As it was already explained, one distinguishing feature of cellular EAs is that the population is structured by establishing some neighboring relations to the individuals in the population. Thus, Neighborhood is a generic class for obtaining the neighbors of a given individual, and therefore any specific neighborhood to be implemented in JCell must inherit from this class. Specifically, the Linear5, Compact9, and Compact13 neighborhoods are implemented in JCell. We should remark that class Neighborhood can be used for defining any neighborhood structure in the population, so it use must not be restricted only to the cellular case. On the other side, in the case of panmictic EAs, as the steady state and the generational ones, there is not any neighborhood, as the parents are chosen among all the individuals belonging to the population (the population is not structured).

Problems in JCell must be implemented as a subclass of Problem. This class is generic enough for allowing the use of problems belonging to such different fields like combinatorial, continuous, integer programming, or even multi-objective optimization. Additionally, the use of constraints is possible in any of the previous problem domains, and the variable values can be restricted to a given interval for each gene of the chromosome. A subclass of Problem must contain every information needed about the considered problem, such as the fitness function, the number of variables, which can be different to the length of the chromosome (since it is possible to code one variable with more than one gene), the number of constraints, the best known fitness value to the problem (only for single-objective optimization), and the interval of allowed values for each gene.

An EvolutionaryAlg object has a pool of Operator objects. This generic class is used for implementing all the EA operators, i.e., the selection method for each of the parents, the recombination and mutation operators, the (optional) local search step, and the replacement policy for the newly generated individuals.

At every generation, the algorithm uses a Statistics object in order to obtain some statistics of the current execution. Computing basic statistics is rarely found in the codes of other authors, but we consider it is an essential step for the work and monitoring of the algorithm. In fact, it is possible to use some kind of statistical descriptors in some of the new algorithmic improvements we compile in this book, e.g., to guide an adaptive search (as we previously saw in Chap. 6).

Additionally, the class EvolutionaryAlg contains an abstract method (called experiment()) which applies the reproductive cycle of the EA. Only extending this abstract method, we define the classes SSGA and GenGA in order to implement steady state (an individual is updated in each generation) and generational (all the individuals are updated in each generation) GAs. Therefore, for extending JCell with any other kind of EA, we only have to define a new class which inherits from EvolutionaryAlg and which implements the reproductive cycle of the new desired EA. Once we have done that, we can use, in our new EA, the already implemented models in EvolutionaryAlg, as the multi-objective, the adaptive or the hierarchical ones, with no effort.

The cGA is implemented with the class CellularGA, which inherits the properties from GenGA (since in both cases the hole population is updated in each generation) and also sets the neighborhood structure inside the population. Particulary, for implementing a cGA in the class CellularGA we need to extend GenGA with:

- **An structured population.** Population does not set any structure between the individuals of the population, what is just a list of individuals without any particular order. Hence, it was necessary to implement a new class for the cellular model, called PopGrid, which inherits from Population and sets an structure on the list of individuals of Population so that a position of a bidimensional grid is assigned to each individual. In our framework it is easy to use other topologies for the population, as the small-world recently proposed for cGAs in some works [101, 203]. For that, it is only necessary to create a class inheriting from Population and setting the desired relation between the individuals.
- **A neighborhood.** The class Neighborhood was implemented for obtaining the neighborhood of a given individual. Nowadays, there exist three classes inheriting from Neighborhood for implementing three of the most known neighborhoods in JCell. These classes are Linear5, Compact9 and Compact13 for the L5, C9 y C13 neighborhoods, respectively (see Chap. 1 for more details about neighborhoods). Adding a new neighborhood is as

easy as creating a new class inheriting from Neighborhood and setting the desired neighborhood structure.

- **Visiting order of the individuals.** As it was said in Chap. 1, there exist some different update policies for cGAs. The use of a synchronous or asynchronous update is established in a variable of class EvolutionaryAlg, whereas the order in which the individuals are visiting is taken from the abstract class CellUpdate. There exist four different update policies in JCell: Line Sweep, Fixed Random Sweep, New Random Sweep, or Uniform Choice. The distinction among these four policies only has sense in the asynchronous case, as in the synchronous one the individuals are updated simultaneously. Therefore, the synchronous cGA can be configured with any of these four policies, excepting the uniform choice so that, as it was commented in Chap. 1, in this policy, maybe, not all the individuals are updated in a giving generation.

There is also a static class called Target which holds a boolean variable, called *maximize*, for storing whether the algorithm is maximizing or minimizing. Additionally, this class is used for making all the comparisons between individuals, deciding in terms of the value of variable *maximize* which one is the best solution to the problem. Thus, all the comparisons between individuals and fitness values are a *black box* for the rest of classes of JCell, so these classes call the same method of Target independently of maximizing or minimizing, if we are holding constraints or not, or even if we are solving a single-objective or multi-objective problem. In this last case, the terms better or worse are not directly applied, the concepts of dominating (considered as better), dominated (worse), and non-dominated (non-worse) individuals are used instead in the comparisons inside *Target*.

11.2 Using JCell

The easiest way for configuring the parameters setting of an EA when working with the JCell framework is to define all the desired parameterization into a simple configuration file. This option, provided by the ReadConf class, has clear advantages, since the user does not need to know the inner details of JCell codification, like the variable names and their data types. Additionally, we can easily change both the parameter settings of our EA and the algorithm itself without the need of compiling the code. These features make our framework accessible to those researchers belonging to any other field, who do not want to discard the usage of a newly optimization technique for their researches, since it is not needed to have any knowledge in programming languages for using JCell. The only thing they have to do is to edit the configuration file for setting the parameterization and then run the `JCell.class` file.

```
##########################################
### JCELL CONFIGURATION FILE           ###
### Asynchronous cGA (FRS) for ECC     ###
##########################################

## Kind of  algorithm
Algorithm = cellular

## Individuals update policy
UpdatePolicy = Asynchronous FRS

## Population shape (x, y)
Population = (20, 20)

## ¿Show a graphical evolution of the population?
ShowDisplay = true

## ¿Show some information during the run?
Verbose = true

## Problem to solve
Problem = problems.Combinatorial.ECC

## Maximum allowed number of evaluations
EvaluationsLimit = 1000000

## Kind of  individuals
Individual = jcell.BinaryIndividual

## Define the neighborhood to use
Neighborhood = jcell.Linear5

## Selection policy
SelectionParent1 = jcell.CenterSelection
SelectionParent2 = jcell.TournamentSelection

## Recombination probability
CrossoverProb = 1.0

## Recombination operator
Crossover = operators.Dpx

## Individual mutation probability
MutationProb = 1.0

## Mutation operator
Mutation = operators.BinaryMutation

## Replacement policy
Replacement = jcell.ReplaceIfNonWorse
```

Fig. 11.2. Configuration file example for solving ECC with an asynchronous cGA

```
######################################
## JCELL CONFIGURATION FILE        ###
## Simple multi-objective ssGA     ###
## for Fonseca problem             ###
######################################

## Kind of algorithm
Algorithm = steady-state

## Population size
Population = 400

## ¿Show some information during the run?
Verbose = true

## Problem to solve
Problem = problems.MO.Fonseca

## Policy for managing the archive
ArchiveManagement = MO.Crowding

## Maximum allowed number of evaluations
EvaluationsLimit = 25000

## Kind of individuals
Individual = jcell.RealIndividual

## Selection policy
SelectionParent1 = jcell.TournamentSelection
SelectionParent2 = jcell.TournamentSelection

## Recombination policy
CrossoverProb = 1.0

## Recombination operator
Crossover = operators.SBX

## Individual mutation probability
MutationProb = 1.0

## Mutation operator
Mutation = operators.NonUniformMutation

## Replacement policy
Replacement = jcell.ReplaceIfNonWorse
```

Fig. 11.3. Configuration file example for solving the multi-objective problem Fonseca with a steady state GA

Therefore, using this kind of configuration file, the functionality and the easy of use of JCell is highly strengthened. This is a really important issue for its success and acceptance by the scientific community. The configuration file is composed of pairs:

$$\text{Parameter Name} = \text{Parameter Value}$$

In Figs. 11.2 and 11.3 we show two examples of configuration files for JCell. Lines starting with # are comments. The file shown in Fig. 11.2 is a typical configuration for a cGA (`Algorithm = cellular`), in which the visiting order of individuals in the breeding loop is defined by an asynchronous fixed

random sweep policy (`UpdatePolicy = Asynchronous FRS`), and the population shape is set to 20×20 individuals (`Population = (20, 20)`). The evolution of the individuals in the population is graphically displayed during the execution (`ShowDisplay = true`), being updated every generation, and JCell will print on the standard output some significative information of the evolution of the population during the run (`Verbose = true`). The problem we want to solve is the combinatorial optimization ECC problem (`Problem = problems.Combinatorial.ECC`), whose class is implemented in the package `problems.Combinatorial` of JCell. The termination condition of the algorithm is to find the optimal solution to the problem (which is given in the class implementing the problem), or to reach a maximum of 1 million fitness function evaluations (`EvaluationsLimit = 1000000`). The chromosome of individuals must be composed by binary genes for solving the selected combinatorial optimization problem (`Individual = jcell.BinaryIndividual`), and the neighborhood structure we want to use is Linear5 (`Neighborhood = jcell.Linear5`), from which one the parents will be selected by a binary tournament (`SelectionParent2 = jcell.TournamentSelection`).

The other parent (which will be selected firstly) is the current individual itself (`SelectionParent1 = jcell.CenterSelection`). The two selected parents are never allowed to be the same individual in JCell. The two point recombination operator (`Crossover = operators.Dpx`) is applied to these two parents with probability 100% (`CrossoverProb = 1.0`), and the newly generated offspring is always mutated (`MutationProb = 1.0`) using the binary mutation operator (`Mutation = operators.BinaryMutation`). The genes of the chromosome are mutated with probability 1/L, being L the length of the chromosome. Finally, the offspring replaces the current individual in the population only if it has a better (or equal) fitness value (`Replacement = jcell.ReplaceIfNonWorse`).

In Fig. 11.3 we can see how easy is to run a steady state multi-objective GA with JCell just by changing a few key parameters in the previously explained configuration file (shown in Fig. 11.2). Just by changing the `Algorithm` key to `steady state` we get a steady state GA. The population in a panmictic algorithm (both steady state and generational) is a pool of individuals, and thus it has no structure nor a concrete shape. Therefore, we directly set the population size in this file instead of its shape (`Population = 400`). The problem we want to tackle is the so called Fonseca problem (`Problem = problems.MO.Foseca`), and the maximum number of fitness function evaluations is set to 25000. Fonseca problem requires a codification of *Double* genes for representing the variables (`Individual = jcell.RealIndividual`), and thus we have to use proper recombination and mutation operators for this kind of chromosome representation: *simulated binary* (`Crossover = operators.SBX`) and non uniform mutation (`Mutation = operators.NonUniformMutation`), respectively. Finally, the neighborhood does not need to be specified (it would be ignored if it were specified) since

Table 11.2. Main configuration parameters of JCell

Parameter	Values	Comments
Algorithm	`cellular`	Cellular GA
	`generational`	Generational GA
	`steady state`	Steady state GA
UpdatePolicy	`Synchronous`	Synchronous cGA
	`Asynchronous LS`	Async. cGA. Line Sweep Policy
	`Asynchronous FRS`	Async. cGA. Fixed Random Sweep Policy
	`Asynchronous NRS`	Async. cGA. New Random Sweep Policy
	`Asynchronous UC`	Async. cGA. Uniform Choice Policy
Population	`(x,y)`	cGA with a $x \times y$ population
	`N`	Panmictic GA with population size of `N`
ShowDisplay	{`true`|`false`}	Graphically show or not the evolution of the population
Verbose	{`true`|`false`}	Show or not the execution traces in the standard output
EvaluationsLimit	`N`	Set maximum allowed evaluations to `N`
Neighborhood	`jcell.Linear5`	The neighborhood of the cGA is L5
	`jcell.Compact9`	The neighborhood of the cGA is C9
	`jcell.Compact13`	The neighborhood of the cGA is C13
HierarchycalPop	{`true`|`false`}	Use or not a hierarchical population
AdaptivePop	`adaptiveCGA.AF`	Adaptive pop. Diversity measure: AF
	`adaptiveCGA.PH`	Adaptive pop. Diversity measure: PH
	`adaptiveCGA.AFPH`	Adaptive pop. Diversity measure: AF+PH
CrossoverProb	`N`	Set recombination probability to `N`
MutationProb	`N`	Set individual mutation probability to `N`
SelectionParent1 and *SelectionParent2*	`jcell.TournamentSelection`	Binary tournament selection
	`jcell.RouleteWheelSelection`	Roulette wheel selection
	`jcell.CenterSelection`	Current individual selection
	`jcell.BestSelection`	Best individual of the neighborhood
	`jcell.LinearRankSelection`	Linear ranking selection
Replacement	`jcell.ReplaceIfNonWorse`	Replace if offspring is not worse
	`jcell.ReplaceIfBetter`	Replace if offspring is better
	`jcell.ReplaceAlways`	Replace always
ArchiveManagement	`CMoEA.Crowding`	The archive is managed using crowding
	`CMoEA.AdaptiveGrid`	The archive is managed using adaptive grid
Crossover	see Table 11.3	Recombination operator
Mutation	see Table 11.4	Mutation operator
Problem	see Table 11.5	Problem to solve

the whole population will be considered as the neighborhood for the steady state GA.

Finally, the main parameters you can use in the configuration file of JCell are briefly explained. In Table 11.2 we show some of the most important configuration parameters for JCell, their possible values and a concise explanation of their meaning. Some examples of these parameters are the kind of the algorithm to use, the mutation and recombination operators and their probabilities, the parents selection operators, or the replacement policy. As it was previously said, for configuring a parameter it is necessary to write in the configuration file, for the desired parameter, the name which appears in the column **Parameter**, a symbol "=" and one of the possible values of the column **Values**.

The different kinds of available recombination and mutation operators in JCell, as well as the implemented problems, are shown in Tables 11.3, 11.4, and 11.5. Tables 11.3 and 11.4 show the available recombination and mutation operators in JCell, respectively, and also the name of the operator they

Table 11.3. Available recombination operators in JCell

Class	Operator name	Comments and parameters
operators.Ax	Asymmetric crossover	For real genotype individuals; `pBiasAX` = 0.5
operators.BLXalpha	Blend crossover	For real genotype individuals; `AlphaBLX` = 0.5
operators.Dpx	Two point crossover	For all kind of individuals
operators.DX	Discrete crossover	For real genotype individuals
operators.ELX	Extended line crossover	For real genotype individuals
operators.Erx	Axis recombination cross.	For permutation of integer chromosomes
operators.FX	Plain crossover	For real genotype individuals
operators.LBGAX	Linear BGA crossover	For real genotype individuals
operators.LX	Linear crossover	For real genotype individuals
operators.OX	Ordered crossover	For permutation of integer chromosomes
operators.PBX	Parents-centric crossover	For real genotype individuals `AlphaPBX` = 0.8
operators.Pmx	Partially matched crossover	For permutation of integer chromosomes
operators.PNX	Normal Parents-centric cross.	For real genotype individuals `LambdaPNX` = 0.35 `RoPNX` = 2.0
operators.Px	Probabilistic crossover	For all kind of individuals `pBiasPX` = 0.5
operators.SBX	Simulated binary crossover	For real genotype individuals `distributionIndexSBX` = 20.0
operators.Spx	One point Crossover	For all kind of individuals
operators.WHX	Wright heuristic crossover	For real genotype individuals

Table 11.4. Available mutation operators in JCell

Class	Operator name	Comments and parameters
operators.BinaryMutation	Binary mutation	For binary genotype individuals
operators.CauchyMutation	Cauchy mutation	For real genotype individuals `deviationCM` = 1.0
operators.FloatUniformMutation	Uniform mutation	For real genotype individuals
operators.FloatNonUniformMutation	Non uniform mutation	For real genotype individuals
operators.GaussianMutation	Gaussian mutation	For real genotype individuals `deviationGM` = 1.0
operators.MuhlenbeinMutation	Mühlenbein mutation	For real genotype individuals
operators.PolynomialMutation	Polynomial mutation	For real genotype individuals `distributionIndexPM` = 20.0

Table 11.5. Problems included in JCell

Class	Comments
problems.MO.Constr_Ex	Multi-objective; Real Codification
problems.MO.Fonseca	Multi-objective; Real Codification
problems.MO.Golinski	Multi-objective; Real Codification
problems.MO.Kursawe	Multi-objective; Real Codification
problems.MO.Schaffer	Multi-objective; Real Codification
problems.Combinatorial.COUNTSAT	Binary Codification
problems.Combinatorial.ECC	Binary Codification
problems.Combinatorial.FMS	Binary Codification; Coded variables with 32 bits
problems.Combinatorial.MAXCUT	Binary Codification; Instances 100, 20.01 and 20.09
problems.Combinatorial.MMDP	Binary Codification
problems.Combinatorial.MTTP	Binary Codification; Instances of 20, 100, and 200 vars.
problems.Combinatorial.OneMax	Binary Codification; 500 variables
problems.Combinatorial.PEAK	Binary Codification
problems.Continuous.Ackley	Real Codification
problems.Continuous.Chebyschev	Real Codification
problems.Continuous.EF10	Real Codification
problems.Continuous.FMS	Real Codification
problems.Continuous.Griewangk	Real Codification
problems.Continuous.Rastrigin	Real Codification
problems.Continuous.Rosenbrock	Real Codification
problems.Continuous.Schwefel	Real Codification
problems.Continuous.Sle	Real Codification
problems.Continuous.Sphere	Real Codification

implement and some comments, as well as specific configuration parameters of each operator, if they have. The default values used for these parameters are those shown in the two tables. Finally, all the available problems are given in Table 11.5 the Java package where they can be found plus some comments such us the codification used. In Appendix A, a complete description of the problems is provided.

11.3 Conclusions

In this chapter we have presented and described JCell, a Java library specially targeted to program cGAs. It is publicly available at `http://neo.lcc.uma.es/software/jcell`. JCell is itself an important result related to the knowledge in this book, as it is a powerful optimization tool for other researchers since they can use it in their studies. Moreover, JCell allows to a specialized researcher in the field of evolutionary algorithm to deepen with very little effort in the studies performed here, to develop new studies based on cellular evolutionary algorithms, or even to export the new proposed ideas in this book to other fields.

Part IV

Applications of cGAs

12

Continuous Optimization

The good is, like Nature, an immense landscape in which Man advances through centuries of exploration.

José Ortega y Gasset (1883 - 1955) – Philosopher

In the previous chapters we have presented several new algorithmic models, and their performance have been tested on academical benchmarks. Now, we turn to demonstrate (in this chapter and the following ones) the high suitability of some of the studied cellular EAs when they are applied to immense landscapes. Thus, we solve here a large benchmark of problems in the continuous domain, while in the following three chapters we tackle with our cGAs three very complex real-world problems belonging to different domains. The main goal in this part of the book is to show that all these ideas are useful in practice, not just nice abstractions.

In Sect. 12.1 of this chapter we briefly justify the study done here. Later, we present in Sect. 12.2 the results obtained, the parametrization of the algorithm, and a comparison of our results versus those of other algorithms belonging to the state of art for the same problems. Finally, we end in Sect. 12.3 by summarizing our main conclusions.

12.1 Introduction

In Chap. 4 we performed a first approximation for the application of cGAs to the continuous domain. In that chapter, a cGA using arithmetic recombination (AX) and a uniform mutation (UM) is applied to three problems in which the variables were represented by real variables (the Rastrigin, Ackley, and Fractal functions), with promising results. The motivation for this chapter is to deepen in the study of the behavior of cGAs when solving real-codified problems. For that we selected a large benchmark commonly used in the literature which is composed of problems with a larger size (and more complexity) than the ones used in Chap. 4. The obtained results are compared to several best known approaches in the literature. One of the main contributions of this chapter is the improvement in the results showed in Chap. 4; moreover, our algorithm, a canonical cGA which implements two very well known recombination and mutation operators, obtains really competitive results, or even better in many cases than the (complex) algorithms compared, belonging to the state of art in continuous optimization.

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_12, © Springer Science+Business Media, LLC 2008

Table 12.1. Parameterization

Population size	144 individuals (12×12)
Neighborhood	NEWS
Parent selection	Binary tournament + binary tournament
Recombination	BLX-α; $\alpha = 0.5$
Recombination probability	$p_c = 1.0$
Mutation	Non-uniform mutation
Mutation probability	$p_m = 1/(2 \cdot n)$
Replacement	Replace_if_Better
Stop condition	800000 fitness evaluations

12.2 Experimentation

In this section we present the results of solving a large benchmark of numerical optimization problems with JCell (our cGA) and compare them to those of other state-of-the-art algorithms. The benchmark used in our study is the same proposed in [127], just for easily comparing our algorithm to others belonging to te state of art in this field. This set of problems is composed of 6 classic academic and well-known functions (Griewangk, Rastrigin, Rosenbrock, Schwefel, Sphere, and ef_{10}), and three additional problems taken from the real world (FMS, SLE, and Chebyschev). All these problems are described in Apendix A. The selected (minimization) problems are defined by really diverse characteristics (linearity, multimodality, scalability, convexity, etc.), and they allow us to compare our canonical cGA to other algorithms in the literature. Therefore, we consider this benchmark wide and diverse enough for sustaining our statements when evaluating the algorithmic approaches (where the compared algorithms are presented).

In this work we use the same parametrization for all the problems. As it can be seen in Table 12.1, we use a population of 144 individuals, in a bidimensional grid of 12×12 cells. The neighborhood used is NEWS, from where two parents are chosen by using *binary tournament* (we force them to be different).

Offsprings are obtained by applying the *blend recombination operator*, BLX-α [86] (with $\alpha = 0.5$), to the two selected parents. The offspring $\mathbf{z}$ generated by BLX-α is composed of genes z_i randomly (uniformly) chosen from the interval $[\min(x_i, y_i) - I \cdot \alpha, \max(x_i, y_i) + I \cdot \alpha]$, where $I = \max(x_i, y_i) - \min(x_i, y_i)$, being x_i and y_i the ith genes of the two parents $\mathbf{x}$ and $\mathbf{y}$, respectively. This recombination is accomplished with probability $p_c = 1.0$, and we apply the *non-uniform mutation* operator to the resulting offspring, considered one of the most suitable ones for real-coded GAs [128]. This mutation operator is applied to the alleles of the offsprings with probability $p_m = 1/(2 \cdot n)$, where n is the size of the problem (length of the chromosome), and mutates them using a non uniform distribution, which step size decreases as the execution advances. Thus, it makes an uniform search in the initial space (exploration stage) and very locally at a larger stage, favoring

Table 12.2. Computational results. Best solutions obtained

Alg.	$\mathbf{f}_{\text{fms}}$	$\mathbf{f}_{\text{Gri}}$	$\mathbf{f}_{\text{Ras}}$	$\mathbf{f}_{\text{Ros}}$	$\mathbf{f}_{\text{Sch}}$	$\mathbf{f}_{\text{Sph}}$	$\mathbf{f}_{\text{ef}_{10}}$	$\mathbf{f}_{\text{SLE}}$	$\mathbf{f}_{\text{Cheb}}$
JCell	4.47*e*-27	**90%**	**80%**	1.33*e*0	**9.28e-161**	**1.72e-158**	**90%**	**26%**	1.28*e*3
$p_m = \frac{2}{n}$	2.95*e*-27	**78%**	**72%**	1.37*e*1	1.57*e*-036	8.14*e*-037	4.38*e*-17	**50%**	1.25*e*3
$p_m = \frac{1}{n}$	9.14*e*-27	**80%**	**68%**	9.87*e*-2	1.63*e*-101	6.89*e*-098	2.20*e*-56	**30%**	1.26*e*3
$\alpha = 0.25$	1.38*e*-04	3.04*e*-4	7.32*e*-5	1.75*e*1	2.31*e*-005	6.44*e*-008	1.58*e*-08	**66%**	1.29*e*3
$\alpha = 0.75$	**3.51e-36**	**48%**	**80%**	**4.538e-3**	1.32*e*-100	4.45*e*-098	1.91*e*-71	**42%**	**1.17e3**
$p_c = 0.50$	4.46*e*-07	**2%**	2.92*e*-7	9.92*e*-1	8.52*e*-030	6.34*e*-023	1.34*e*-06	**14%**	1.23*e*3
$p_c = 0.75$	1.25*e*-18	**60%**	**40%**	1.39*e*0	4.51*e*-134	1.30*e*-130	6.60*e*-80	**50%**	1.28*e*3

local exploitation. Finally, we use an elitist replacement policy (*replace if better*), in which the offspring replaces the current individual in the population of the next generation only if it is better (lower fitness value). The algorithm stops after performing 800000 evaluations, the same condition used in [127].

For the experiments in this paper, 50 independent runs were made for every problem and algorithm. In order to obtain statistical significance in our comparisons we perform ANOVA or Kruskal-Wallis tests on the results (depending on whether the data are normally distributed or not, respectively). The normality of the data distribution is checked by using the Kolmogorov-Smirnov test. Matlab(c) was used for this statistical significance study.

12.2.1 Tuning the Algorithm

We proceed in this section to adjust the values for some parameters of the cGA in order to improve the behavior of the algorithm as much as possible. The base parameterization is that described in Table 12.1 (algorithm JCell), and we test the algorithm with two different mutation probabilities ($p_m = 2/n$ and $p_m = 1/n$), two different values for α ($\alpha = 0.25$ and $\alpha = 0.75$), and two recombination probabilities ($p_c = 0.5$ and $p_c = 0.75$). The best values obtained for every problem with the 7 cGAs are shown in Table 12.2. If the global optimum (the exact 0.0 value, with no error) has been located throughout some runs, we will present the percentage of runs in which this happens (in this case, a '%' symbol appears just after the figure in the table). Additionally, the average obtained results, together with their typical deviations, are given in Table 12.3. The best values in the two tables for each problem are **bolded**.

Regarding Table 12.2, we can see that JCell (using the parameterization of Table 12.1) is the most accurate of the studied settings since it obtains the best results for 6 out of the 9 tested problems. Additionally, JCell also obtains the best overall behavior in terms of the average fitness values found in the 50 runs (4 out of the 9 problems), as it can be seen in Table 12.3. In the other 5 problems, the results reported by JCell have the same order of magnitude as the best value of the compared cGAs, with the exception of f_{Sle}. There exists statistical confidence in the results of the compared cGAs for all the problems.

Table 12.3. Computational results. Average solutions

Alg.	f_{fms}	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{ef_{10}}$	f_{SLE}	f_{Cheb}
JCell	9.06e0 ±6.65e0	**2.38e-3** ±4.75e-3	2.73e-5 ±9.30e-6	1.78e1 ±1.09e1	**6.58e-158** ±2.18e-157	**1.50e-154** ±5.47e-154	**1.37e-3** ±5.80e-3	4.32e-10 ±2.36e-09	1.39e3 ±4.80e1
$p_m = \frac{2}{n}$	**2.11e0** ±4.57e0	3.04e-3 ±6.12e-3	9.11e-5 ±2.82e-4	2.37e1 ±1.38e1	7.52e-035 ±1.12e-034	2.17e-034 ±5.13e-034	1.14e-2 ±2.97e-2	1.57e-05 ±1.11e-04	1.41e3 ±6.14e1
$p_m = \frac{1}{n}$	4.07e0 ±5.79e0	3.53e-3 ±5.40e-3	**7.31e-6** ±2.97e-5	2.14e1 ±1.36e1	2.30e-098 ±9.13e-098	1.22e-095 ±4.25e-095	4.83e-3 ±1.15e-2	4.84e-10 ±3.42e-09	1.40e3 ±5.84e1
$\alpha = 0.25$	7.66e0 ±6.49e0	1.54e-2 ±1.37e-2	4.92e-4 ±5.04e-4	2.27e1 ±7.45e0	9.67e-002 ±9.67e-002	2.09e-006 ±1.72e-006	9.68e-2 ±1.05e-1	2.38e-05 ±1.69e-04	1.42e3 ±5.71e1
$\alpha = 0.75$	7.75e0 ±7e0	8.78e-3 ±1.04e-2	3.17e-5 ±9.90e-5	**1.13e1** ±1.35e1	2.87e-097 ±8.31e-097	4.87e-095 ±2.68e-094	3.85e-3 ±9.98e-3	**1.78e-14** ±1.68e-14	**1.38e3** ±6.65e1
$p_c = 0.50$	7.98e0 ±7.01e0	8.96e-3 ±1.28e-2	1.01e-4 ±1.34e-4	3.30e1 ±2.67e1	1.84e-003 ±1.07e-002	2.73e-008 ±8.18e-008	2.96e-1 ±4.96e-1	4.47e-05 ±2.96e-04	1.39e3 ±6.15e1
p_c =0.75	9.41e0 ±7.46e0	6.81e-3 ±9.7e-3	2.85e-5 ±8.53e-5	2.67e1 ±2.07e1	2.16e-010 ±1.53e-009	2.65e-014 ±1.87e-013	3.38e-2 ±5.81e-2	1.49e-07 ±8.85e-07	1.39e3 ±4.69e1
Test	+	+	+	+	+	+	+	+	+

Table 12.4. Computational results. Average execution times (seconds)

Alg.	f_{fms}	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{ef_{10}}$	f_{SLE}	f_{Cheb}
JCell	**132.79** ±2.90	**7.39** ±7.70	**9.63** ±5.46	24.87 ±0.22	**22.43** ±0.20	18.31 ±0.18	24.93 ±0.37	10.41 ±5.63	540.96 ±1.05
$p_m = \frac{2}{n}$	140.17 ±1.69	15.97 ±4.45	19.08 ±2.65	**24.69** ±0.34	23.49 ±0.16	19.26 ±0.16	27.16 ±0.13	7.35 ±4.14	**484.21** ±0.82
$p_m = \frac{1}{n}$	137.74 ±2.71	9.87 ±0.18	11.94 ±0.11	29.61 ±0.21	26.97 ±0.25	17.70 ±0.14	27.13 ±0.25	7.70 ±4.40	565.67 ±1.78
$\alpha = 0.75$	135.68 ±3.58	11.84 ±8.13	10.60 ±6.06	29.73 ±0.54	25.84 ±0.14	17.47 ±0.19	26.40 ±0.07	6.88 ±3.96	564.40 ±1.89
$p_c = 0.50$	135.57 ±3.35	17.86 ±2.75	17.52 ±0.10	27.03 ±0.41	23.47 ±0.11	**14.97** ±0.11	**23.72** ±0.21	7.40 ±2.41	562.64 ±1.33
p_c =0.75	139.05 ±2.97	11.11 ±8.16	14.63 ±5.40	28.24 ±0.47	25.14 ±0.14	16.21 ±0.22	24.22 ±0.64	7.07 ±4.50	563.36 ±2.34
Test	+	+	+	+	+	+	+	+	+

From Tables 12.2 and 12.3 we can see that changing the probabilities of recombination and mutation, and the value of α from the parameterization of the algorithm proposed in Table 12.1 does not lead us to improve the behavior of the algorithm, in general. The most noticeable improvements are those of f_{Ros} and f_{Sle} when using $\alpha = 0.75$, and f_{fms} when increasing the mutation probability to $2/n$. From these three cases, only f_{fms} with $p_m = 2/n$ is better than JCell with statistical confidence. Hence, we conclude that JCell is the best one of the proposed algorithms and thus we select it for our comparisons with other approaches in the literature in the next section.

In order to make a deeper study of the proposed algorithms, we also include a comparison on the average run times (in seconds) performed by the algorithms for the proposed problems in Table 12.4 (the best –lower– result for each problem is **bolded**). As it can be seen, there exist statistically significant differences for every problem in the reported values. Thus, JCell is the fastest of the compared algorithms for 4 out of the nine problems (statistical differences were found for f_{fms}, f_{Gri}, and f_{Sch}), while there exist algorithms faster than JCell with statistical significance in only 3 out of all the studied problems (f_{Sph}, $f_{\text{ef}_{10}}$, and f_{Sle}). The most important differences in the values

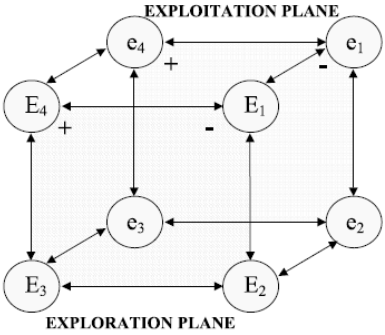


Fig. 12.1. Structure of the gradual distributed real-coded GA

reported in Table 12.4 are due to the distinct hit rates obtained, since the algorithm stops before reaching 800000 evaluations when the optimum is found. As an example, JCell is between 2 and 3 times faster than the algorithm with $\alpha = 0.25$ when the optimal solution is found by the two algorithms, what is contradictory to the results of Table 12.4, reporting 10.41 seconds for JCell and 5.65 seconds for the other algorithm (with $\alpha = 0.25$) in average. The reason is that the latter finds the optimum in 66% of the runs while the hit rate of the former is only 26% (see Table 12.2).

12.2.2 Comparison with Other Algorithms

In this section we compare JCell with other algorithms belonging to the state of the art in continuous optimization. These algorithms are the gradual distributed real-coded GAs proposed in the work of Herrera and Lozano [127]. The problems studied in that work are the same as in our benchmark, as well as the dimension of the search space used for each problem. (10 variables for $f_{\text{ef}_{10}}$ and 25 for the rest of the problems.) Additionally, the termination condition of these compared algorithms is the same used in this work: achieving 800000 fitness function evaluations or finding the optimal solution.

The three compared algorithms [127] are heterogeneous distributed real-coded GAs differing in the recombination operator used. They are based on an hypercube topology with three dimensions and there is a panmictic population of 20 individuals in each vertex of the cube (see Fig. 12.1). Each subpopulation implements a different parameterization of the crossover operator such that those populations in the front side of the cube are devoted to exploration, while exploitation is enhanced in the crossover operators of the rear side populations. With a given frequency, there exist migrations of individuals between populations belonging to the same sides of the cube. The three heterogeneous distributed GAs presented in [127] are GD-FCB, which use the *fuzzy connectives based crossover operator* (FCB) [126], GD-BLX, implementing the *blend crossover operator* (BLX-α) [86], and GD-EFR, using the *extended fuzzy recombination operator* (EFR) [255]. The reader is referred to [127] for a complete description of this algorithmic model.

Table 12.5. Comparison with other approaches. Best solution found

Algorithm	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{\text{ef}_{10}}$	f_{fms}	f_{SLE}	f_{Cheb}
JCell	**90%**	80%	1.3e0	**9.3e-161**	**1.7e-158**	**90%**	4.5e-27	**26%**	1.3e3
GD-FCB	4e-11	9e-12	**3e-5**	6e-001	4e-014	8e-04	7e-26	3e0	4e1
GD-BLX	60.0%	**100%**	2e1	7e-008	8e-058	5e-48	**66.7%**	2e0	**1e1**
GD-EFR	53.3%	**100%**	1e1	2e-007	8e-051	6e-47	43.3%	9e-1	**1e1**

Table 12.6. Comparison with other approaches. Average of solutions found

Algoritmo	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{\text{ef}_{10}}$	f_{fms}	f_{SLE}	f_{Cheb}
JCell	**2.4e-3**	2.7e-05	1.8e1	**6.6e-158**	**1.5e-154**	1.4e-03	9.1e0	**4.3e-10**	1.4e3
GD-FCB	2e-2	4e-11	**9e0**	4e000	2e-013	2e-03	1e1	4e01	**2e2**
GD-BLX	5e-3	**0e00**	2e1	2e-006	8e-053	**6e-36**	**3e0**	3e01	**2e2**
GD-EFR	7e-3	**0e00**	2e1	4e-006	4e-047	9e-35	6e0	2e01	**2e2**

The results of all these algorithms, together with the results of JCell, are given in Tables 12.5 (best found solution) and 12.6 (average of the values found by the algorithms).

If we pay attention to the best solutions found by the algorithm (Table 12.5), we can see that JCell obtains the best results in 5 out of the 9 problems. Additionally, in case of $f_{\text{ef}_{10}}$, JCell finds the optimal solution in the 90% of the runs while the other three approaches are not able to find the optimal solution in any run (the best approach is 5E-48 for GD-BLX). Also for f_{SLE} JCell is the only one that finds the optimal solution to the problem, and it finds it in the 26% of the runs. The difference between JCell and the best compared algorithm is small in the case of f_{Ras}, so that it finds the optimum in the 80% of the solutions against the 100% of GD-BLX and GD-EFR.

Regarding the average solutions found, JCell outperforms the other approaches in 4 out of the 9 functions of Table 12.6, with the important differences of more than 150 orders of magnitude for f_{Sch} and f_{Sph} (in mean and in best solutions) and 11 orders of magnitude for f_{SLE} with respect to the best values reported by the other algorithms. Additionally, we do not consider important the existing differences between JCell, GD-BLX, and GD-EFR for f_{Ras} since JCell finds the local optimum in 80% of the runs (versus 100% for GD-BLX and GD-FER). In the case of $f_{\text{ef}_{10}}$, the bad average value reported by JCell for this problem is because it gets stuck in local optima (value about 10^{-2}) in 10% of the runs. Finally, the average solution found by JCell for f_{Ros} is almost in the same order of magnitude than that of the best algorithm, GD-FCB.

According to the proposed problems based on the real world, in the case of f_{fms}, the average result thrown by all the algorithms is in the same order of magnitude, and the best solution found by JCell is close to the optimum (4.5E-27), although GD-BLX and GD-EFR found the optimal solution in the 66.7% and 43.3% of the runs, respectively. Finally, f_{Cheb} is the unique problem for which JCell is the worst of the compared algorithms both in terms of best and average solutions found. However, the differences between JCell and the other algorithms are only one order of magnitude in the average of the solutions found.

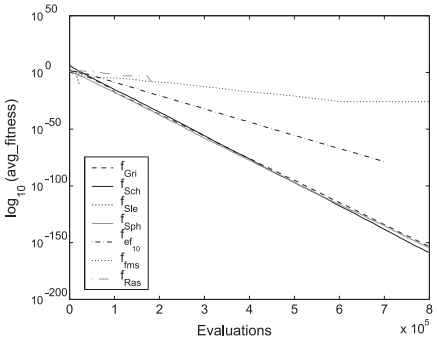


Fig. 12.2. Evolution of the average fitness value for f_{Gri}, f_{Sch}, f_{SLE}, f_{Sph}, $f_{\text{ef}_{10}}$, f_{fms} y f_{Ras}

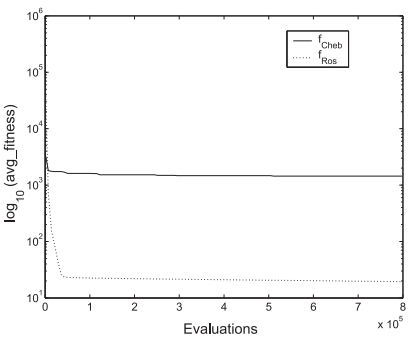


Fig. 12.3. Evolution of the average fitness value for f_{Cheb}, and f_{Ros}

Finally, we can summarize these results concluding that JCell clearly outperforms the three heterogenous distributed approaches in 5 out of the 9 tested instances, improving the current state of the art, and it is very close to the results of the best algorithms in the other problems. Maybe, the exception is f_{Cheb}, for which JCell reports the worst results of the compared algorithms, although the differences are only of one order of magnitude.

At this point, we would like to emphasize the simplicity of our approach versus the three distributed ones. We are using in this work a simple canonical cGA for solving the problems, while the compared algorithms are heterogeneous distributed GAs composed of 8 different sub-populations having each of them distinct parameterizations, with the consequent increment in the parameters needed by the algorithms. Moreover, our approach uses exactly the same recombination and mutation operators implemented in GD-BLX, so the difference in the performance of the two algorithms is intrinsic to the algorithmic model. Hence, the exploration/exploitation tradeoff accomplished by the canonical cGA on the population, achieved by simply introducing the concept of neighborhood in a structured grid of individuals, reports similar results, and in several cases even better, with respect to the complex hypercube model that Herrera and Lozano propose in [127].

In Figs. 12.2 and 12.3 we plot the evolution of the average fitness value of the individuals in the population during typical executions of JCell for all the studied problems. Note that this average fitness value (ordinate axis) is in logarithmic scale. For the functions plotted in Fig. 12.2, it can be seen in general a fast approximation to the best solution of the problem during the execution, meaning a progressive convergence of the population. Among these problems, it stands out the difficulty of f_{fms}, which reports the slowest convergence. After increasing the maximum number of allowed evaluations highly enough, we have checked that this convergence is maintained until the optimum is found in most runs for the functions in Fig. 12.2. The exception is f_{fms}, which falls into local optima from which the algorithm can not escape in several runs.

Among those studied in this work, the most difficult problems for JCell are f_{Cheb}, and f_{Ros}. The two problems are plotted in Fig. 12.3. It can be seen how the algorithm experiments a really fast convergence at the beginning of the search (notice that the average fitness value is plotted in logarithmic scale) but, after that, it gets stuck in local optima, the population diversity is quickly lost, and thus there are no more important improvements in the solutions.

12.3 Conclusions

We have proposed in this chapter a new approach for solving problems in a continuous search space. Our proposal consists of a simple canonical cGA using two well-known recombination and mutation operators in the field of real-coded GAs.

The results reported in this first approach are really promising, since JCell outperformed other existing complex heterogeneous distributed algorithms belonging to the state of the art. Additionally, one of these algorithms (GD-BLX) implements exactly the same recombination and mutation operators we use in JCell, what enhances the exploration/exploitation tradeoff that cGAs perform into the population simply by using the concept of overlapped neighborhoods. In the case of the three compared heterogeneous GAs, this tradeoff between exploration and exploitation is accomplished by distributing the population in small sub-populations with different parameterizations in a hypercube, and restricting the interchange of individuals (migration) among sub-populations located in the same side of the hypercube. Looking at the reported results, we can conclude that the complex hypercube model of the three compared algorithms does not perform a better exploration/exploitation tradeoff than JCell, which is based on concepts much simpler and requires a lower number of parameters to be set (each subpopulation in the hypercube implements a different parameterization).

13

Logistics: The Vehicle Routing Problem

Competition is the keen cutting edge of business, always shaving away at costs.

Henry Ford (1863 - 1947) – Businessman

Transportation plays an important role in logistic tasks of many companies since it usually accounts for a high percentage of the cost added to goods. Therefore, the use of computerized methods in transportation often results in significant savings of up to 20% of the total costs (see Chap. 1 in [248]).

A distinguished problem in the field of transportation consists in finding the optimal routes for a fleet of vehicles which serve a set of clients. In this problem, an arbitrary set of clients must receive goods from a central depot. This general scenario presents many chances for defining (related) problem scenarios, for example: determining the optimal number of vehicles, finding the shortest routes, and so on, being all of them subject to many restrictions such as vehicle capacity, time windows for deliveries, etc. This variety of scenarios leads to a plethora of problem variants in practice. Some reference case studies where the application of vehicle routing algorithms has led to substantial cost savings can be found in Chaps. 10 and 14 in [248].

As stated before, the vehicle routing problem (VRP) [56] consists of delivering goods to a set of customers with known demands through minimum-cost vehicle routes. These routes must start and finish at the depot, as can be seen in Figs. 13.1a and 13.1b (a detailed description of the problem is available in Sect. 13.1).

The VRP is a very important source for problems, since solving it is equivalent to solving multiple TSP problems at once [96]. Due to the difficulty of this problem (NP-hard) and because of its many industrial applications, it has been largely studied both theoretically and in practice [47]. There is a large number of extensions to the canonical VRP. One basic extension is known as the capacitated VRP –CVRP–, in which vehicles have fixed capacities of a single commodity. Many different variants can be constructed from CVRP; some of the most important ones [248] are those including time windows restrictions –VRPTW– (customers must be supplied following a certain time schedule), pickups and deliveries –VRPPD– (customers will require goods to be either delivered or picked up), and backhauls –VRPB– (like VRPPD, but deliveries must be completed before any pickups are made).

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_13, © Springer Science+Business Media, LLC 2008

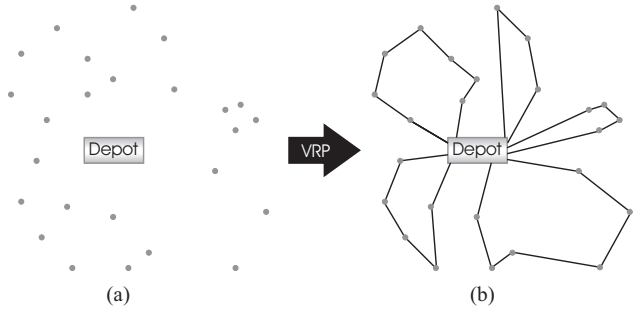


Fig. 13.1. The vehicle routing problem consists in serving a set of geographically distributed customers (points) from a depot (a) using the minimum cost routes (b)

In fact, there are many more extensions for this problem, like the use of multiple depots (MDVRP), split deliveries (SDVRP), stochastic variables (SVRP), or periodic scheduling (PVRP). The reader can find a public web site with all of them, the latest best-known solutions, papers and related stuff in our web site [71].

We consider in this chapter the capacitated vehicle routing problem (CVRP), in which a fixed fleet of delivery vehicles of the same capacity must service known customer demands for a single commodity from a common depot at minimum transit costs. The CVRP has been studied in a large number of separate works in the literature, but (to our knowledge) no work addresses all the available benchmarks together, since it means solving 160 different instances. We use such a large set of instances to test the behavior of our algorithm in many different scenarios in order to give a deep analysis of it and a general view of this problem not biased by any ad hoc selection of individual instances. The included instances are characterized by many different features: instances from real world, theoretically motivated ones, clustered, non-clustered, with homogeneous or heterogeneous demands on customers, with the existence of drop times or not, etc.

In recent VRP history, there has been a steady evolution in the quality of the solution methodologies used, borrowed both from the exact and the heuristic fields of research. However, due to the difficulty of the problem, no known exact method is capable of *consistently* solving to optimality instances involving more than 50 customers [109, 248]. In fact, it is also clear that, as for most complex problems, non-customized heuristics would not compete in terms of the solution quality with present techniques like the ones described in [52, 248]. Moreover, the power of exploration of some modern techniques like genetic algorithms or ant systems is not yet fully explored, specially when combined with an effective local search step. All these considerations allow us fruitful research to refine solutions to optimality. Particularly, we present in this chapter an efficient cellular memetic algorithm (cMA) obtained after hybridizing a canonical cGA with specialized recombination and mutation operators and with a local search algorithm for solving CVRP.

The contribution of this work is then to define a powerful yet simple cMA capable of competing with the best known approaches for solving CVRP in terms of accuracy (final cost) and computational effort (the number of evaluations made). For that purpose, we test our algorithm over the mentioned large selection of instances (160), which will allow us to guarantee deep and meaningful conclusions. Besides, we compare our results against the best existing ones in the literature, some of which we even improve. In [11] the reader can find a seminal work with a comparison between our algorithm and some other known heuristics for a reduced set of 8 instances. In that work, we showed the advantages of embedding local search techniques into a cGA for solving CVRP, since our hybrid cGA was the best algorithm out of all those compared in terms of accuracy and time. Cellular GAs represent a paradigm much simpler to comprehend and customize than others such as tabu search (TS) [97, 249] and similar (very specialized or very abstract) algorithms [37, 207]. This is an important point too, since the greatest emphasis on simplicity and flexibility is nowadays a must in research to achieve widely useful contributions [52].

The chapter is organized in the following manner. In Sect. 13.1 we define CVRP. The proposed cMA is thoroughly described in Sect. 13.2. Section 13.3 presents the results of our tests, comparing them with the best-known values in the literature. After that, we present the new best-known solutions found in our studies for CVRP in Sect. 13.4. Finally, our conclusions and future lines of research are discussed in Sect. 13.5.

13.1 The Vehicle Routing Problem

The VRP can be defined as an integer programming problem which falls into the category of NP-hard problems [161]. Among the different variants of VRP we work here with the Capacitated VRP (CVRP), in which every vehicle has a uniform capacity of a single commodity. The CVRP is defined on an undirected graph $G = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V} = \{v_0, v_1, \ldots, v_n\}$ is a vertex set and $\mathbf{E} = \{(v_i, v_j)/v_i, v_j \in \mathbf{V}, i < j\}$ is an edge set.

Vertex v_0 stands for the *depot*, and it is from where m identical vehicles of capacity Q must serve all the *cities* or *customers*, represented by the set of n vertices $\{v_1, \ldots, v_n\}$. We define on E a non-negative *cost*, *distance* or *travel time* matrix $C = (c_{ij})$ between customers v_i and v_j. Each customer v_i has non-negative demand of goods q_i and drop time δ_i (time needed to unload all goods). Let $\mathbf{V_1}, \ldots, \mathbf{V_m}$ be a partition of $\mathbf{V}$, a route $\mathbf{R_i}$ is a permutation of the customers in $\mathbf{V_i}$ specifying the order of visiting them, starting and finishing at the depot v_0. The cost of a given route $\mathbf{R_i} = \{v_{i_0}, v_{i_1}, \ldots, v_{i_{k+1}}\}$, where $v_{i_j} \in \mathbf{V}$ and $v_{i_0} = v_{i_{k+1}} = 0$ (0 denotes the depot), is given by:

$$\text{Cost}(\mathbf{R_i}) = \sum_{j=0}^{k} c_{j,j+1} + \sum_{j=0}^{k} \delta_j \ , \tag{13.1}$$

and the cost of the problem solution ($\mathbf{S}$) is:

$$F_{\text{CVRP}}(\mathbf{S}) = \sum_{i=1}^{m} \text{Cost}(\mathbf{R_i}) \ . \tag{13.2}$$

CVRP consists in determining a set of m routes (i) of minimum total cost –as it is specified in Eq. 13.2–; (ii) starting and ending at the depot v_0; and such that (iii) each customer is visited exactly once by exactly one vehicle; subject to the restrictions that (iv) the total demand of any route does not exceed Q ($\sum_{v_j \in \mathbf{R_i}} q_j \leq Q$); and (v) the total duration of any route is not larger than a preset bound D ($\text{Cost}(\mathbf{R_i}) \leq D$). All vehicles have the same capacity and carry a single kind of commodity. The number of vehicles is either an input value or a decision variable. In this study, the length of routes is minimized independently of the number of vehicles used.

It is clear from our description that the VRP is closely related to two difficult combinatorial problems. On the one hand, we can get an instance of the *Multiple Travelling Salesman Problem* (MTSP) just by setting $Q = \infty$. An MTSP instance can be transformed into a TSP instance by adjoining to the graph $k - 1$ additional copies of node 0 (depot) and its incident edges (there are no edges among the k depot nodes). On the other hand, the question of whether there exists a feasible solution for a given instance of the VRP is an instance of the *Bin Packing Problem* (BPP). So the VRP is extremely difficult to solve in practice because of the interplay of these two underlying difficult models (TSP and BPP). In fact, the largest solvable instances of the VRP are two orders of magnitude smaller than those of the TSP [208].

13.2 Proposed Algorithms

In this section we present a detailed description of the operators we implemented in our algorithm (for a complete description of the algorithm itself the reader should refer to Chap. 1). In Alg. 13.1 the pseudo-code of JCell2o1i, the cMA we propose in this study, is given. Basically, we use a simple cGA highly hybridized with specific recombination and mutation operators, and also with an added local post-optimization step (line 12). This local search method lies in applying *1-Interchange*, and then improving the best obtained solution using *2-Opt*. These two methods are well-known local search optimization algorithms in the literature (they are described in detail in Sect. 13.2.4).

The fitness value assigned to individuals is computed as follows [11, 97]:

$$f(\mathbf{S}) = F_{\text{CVRP}}(\mathbf{S}) + \lambda \cdot \text{overcap}(\mathbf{S}) + \mu \cdot \text{overtm}(\mathbf{S}) \ , \tag{13.3}$$

$$f_{\text{eval}}(\mathbf{S}) = f_{\max} - f(\mathbf{S}) \ . \tag{13.4}$$

Algorithm 13.1 Pseudo-code of JCell2o1i

```
1. proc Evolve(cma)        //Algorithm parameters in 'cma'
2. GenerateInitialPopulation(cma.pop);
3. Evaluation(cma.pop);
4. while !StopCondition() do
5.    for individual ← 1 to cma.popSize do
6.       neighbors ← GetNeighbors(cma,position(individual));
7.       parents ← Select(neighbors);
8.       offspring ← Recombination(cma.Pc,parents);
9.       offspring ← Mutation(cma.Pm,offspring);
10.      offspring ← LocalSearch(cma.Pl,offspring);
11.      Evaluation(offspring);
12.      InsertIfNotWorse(position(x,y),offspring,cma,aux_pop);
13.   end for
14.   cma.pop ← aux_pop;
15.   UpdateStatistics(cma);
16. end while
17. end proc Evolve;
```

The objective of our algorithm is to maximize $f_{\text{eval}}(\mathbf{S})$ (Eq. 13.4) by minimizing $f(\mathbf{S})$ (Eq. 13.3). The value $f_{\max}$ must be larger or equal with respect to that of the worst feasible solution for the problem. Function $f(\mathbf{S})$ is computed by adding the total costs of all the routes ($F_{\text{CVRP}}(\mathbf{S})$ –see Eq. 13.2–), and penalizes the fitness value only in the case that the capacity of any vehicle and/or the time of any route are exceeded. Functions 'overcap($\mathbf{S}$)' and 'overtm($\mathbf{S}$)' return the excess in capacity and time of the solution (respectively) with respect to the maximum allowed value for each route. The values returned by 'overcap($\mathbf{S}$)' and 'overtm($\mathbf{S}$)' are weighted by factors λ and μ, respectively. In this work we have used $\lambda = \mu = 1000$ [78].

In Sects. 13.2.1 to 13.2.4 we proceed to explain in detail the main features that characterize our algorithm (JCell2o1i). The algorithm itself can be applied with all the mentioned operations and also with only some of them to analyze their separate contribution to the performance of the search, as it was previously done for this algorithm in [14].

13.2.1 Problem Representation

In a GA, each individual represents one candidate solution. A candidate solution to an instance of the CVRP must specify the number of vehicles required, the allocation of the demands to these vehicles, and also the delivery order of each route. We adopted a representation consisting of a permutation of integer numbers. Each permutation will contain both customers and route splitters (delimiting different routes), so we will use a permutation of numbers $[0 \ldots n-1]$ with length $n = c + k$ for representing a solution for the CVRP with c customers and a maximum of $k+1$ possible routes. Customers

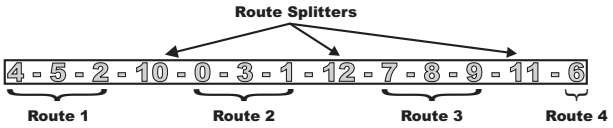


Fig. 13.2. Individual representing a solution for 10 customers and 4 vehicles

are represented with numbers $[0 \dots c-1]$, while route splitters belong to the range $[c \dots n-1]$. Note that due to the nature of the chromosome (permutation of integer numbers) route splitters must be different numbers, although it should be possible to use the same number for designating route splitters when using other possible chromosome configuration.

Each route is composed of the customers between two route splitters in the individual. For example, in Fig. 13.2 we plot an individual representing a possible solution for a hypothetical CVRP instance with 10 customers using at most 4 vehicles. Values $[0, \dots, 9]$ represent the customers while $[10, \dots, 12]$ are the route splitters. **Route 1** begins at the depot, visits customers 4–5–2 (in that order), and returns to the depot. **Route 2** goes from the depot to customers 0–3–1 and returns. The vehicle of **Route 3** starts at the depot and visits customers 7–8–9. Finally, in **Route 4**, only customer 6 is visited from the depot. Empty routes are allowed in this representation simply by placing two route splitters contiguously without any client between them.

13.2.2 Recombination

Recombination is used in GAs as an operator for combining parts in two (or more) patterns in order to transmit (hopefully) good building blocks in them to their offspring. The recombination operator we use is the edge recombination operator (ERX) [262], since it has been largely reported as the most appropriate for permutations compared to other general operators like order crossover (OX) or partially matched crossover (PMX). ERX builds an offspring by preserving edges from its two parents. For that, an *edge list* is used. This list contains, for each city, the edges of the parents that start or finish in it (see Fig. 13.3).

After constructing the edge list of the two parents, the ERX algorithm builds one child solution by proceeding as follows. The first gene of the offspring is chosen from between the first one of both parents. Specifically, the gene having a lower number of edges is selected. In the case of a tie, the first gene of the first parent will be chosen. The other genes are chosen by taking from among their neighbors the one with the shortest edge list. Ties are broken by choosing the first city found that fulfill this shortest list criterion. Once a gene is selected, it is removed from the edge list.

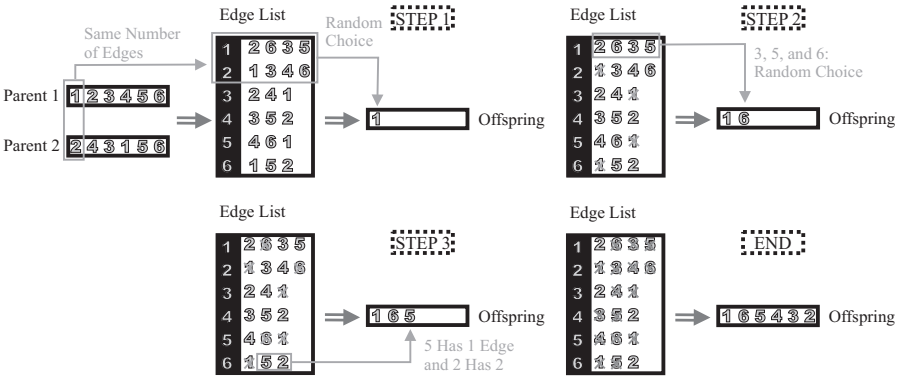


Fig. 13.3. Edge recombination operator (ERX)

13.2.3 Mutation

The mutation operator we use in our algorithm will play an important role during the evolution since it is in charge of introducing a considerable degree of diversity in each generation, counteracting in this way the strong selective pressure which is a result of the local search method we plan to use. The mutation consists of applying *Insertion*, *Swap* or *Inversion* operations to each gene with equal probability (see Alg. 13.2).

These three mutation operators (see Fig. 13.4) are well-known methods found in the literature, and typically applied sooner than later in routing problems. Our idea here is to merge they three in a new *combined* operator. The *Insertion* operator [89] selects a gene (either customer or route splitter) and inserts it in another randomly selected place of the same individual. *Swap* [36] lies in randomly selecting two genes in a solution and exchanging them. Finally, *Inversion* [134] reverses the visiting order of the genes between two randomly selected points of the permuta-

Algorithm 13.2 The mutation algorithm

```
1.  proc Mutation(pm, ind)
    // 'pm' is the mutation probability, and 'ind' is the individual to mutate
2.  for i ← 1 to ind.length() do
3.    if rand0to1()<pm then
4.      r = rand0to1();
5.      if r < 0.33 then
6.        ind.Inversion(i, randomInt(ind.length()));
7.      else if r > 0.66 then
8.        ind.Insertion(i, randomInt(ind.length()));
9.      else
10.       ind.Swap(i, randomInt(ind.length()));
11.     end if
12.   end if
13. end for
14. end proc Mutation;
```

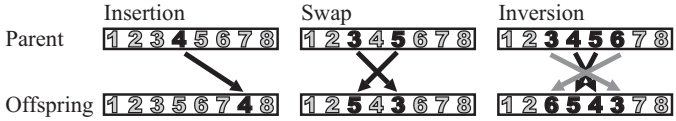


Fig. 13.4. The three different used mutations

tion. Note that the induced changes might occur in an intra or inter-route way in all the three operators. Formally stated, given a potential solution $S = \{s_1, \ldots, s_{p-1}, s_p, s_{p+1}, \ldots, s_{q-1}, s_q, s_{q+1}, \ldots, s_n\}$, where p and q are randomly selected indexes, and n is the sum of the number of customers plus the number of route splitters ($n = c + k$), then the new solution S' obtained after applying each of the different proposed mechanisms is shown below:

$$\text{Insertion}: \quad S' = \{s_1, \ldots, s_{p-1}, s_{p+1}, \ldots, s_{q-1}, s_q, s_p, s_{q+1}, \ldots, s_n\}\,, \tag{13.5}$$

$$\text{Swap}: \quad S' = \{s_1, \ldots, s_{p-1}, s_q, s_{p+1}, \ldots, s_{q-1}, s_p, s_{q+1}, \ldots, s_n\}\,, \tag{13.6}$$

$$\text{Inversion}: \quad S' = \{s_1, \ldots, s_{p-1}, s_q, s_{q-1}, \ldots, s_{p+1}, s_p, s_{q+1}, \ldots, s_n\}\,. \tag{13.7}$$

13.2.4 Local Search

It is very clear after all the existing literature on VRP that the utilization of a local search method is almost mandatory to achieve results of high quality [11, 37, 217]. This is why we envision from the beginning the application of two of the most successful techniques in recent years. In effect, we will add a local refining step to some of our algorithms consisting in applying 2-Opt [55] and 1-Interchange [200] local optimization to every individual.

On the one hand, the 2-Opt simple local search method works inside each route. It randomly selects two non-adjacent edges (i.e., (a, b) and (c, d)) of a single route, deletes them, thus breaking the tour into two parts, and then reconnects those parts in the other possible way: (a, c) and (b, d) (Fig. 13.5a). Hence, given a route $R = \{r_1, \ldots, r_a, r_b, \ldots, r_c, r_d, \ldots, r_n\}$, being (r_a, r_b) and (r_c, r_d) two randomly selected non-adjacent edges, the new route R' obtained after applying the 2-Opt method to the two considered edges will be $R' = \{r_1, \ldots, r_a, r_c, \ldots, r_b, r_d, \ldots, r_n\}$. 2-Opt is similar to the *inversion* mutation operator with the exception that 2-Opt is applied into routes, while *inversion* can affect one or more routes.

On the other hand, the λ-Interchange local optimization method that we use is based on the analysis of all the possible combinations for up to λ customers between sets of routes (Fig. 13.5b). Hence, this method results in customers either being shifted from one route to another, or being exchanged between routes. The mechanism can be described as follows. A solution to the problem is represented by a set of routes $S = \{R_1, \ldots, R_p, \ldots, R_q, \ldots, R_k\}$, where R_i is the set of customers serviced in route i. New neighboring solutions can be obtained after applying λ-Interchange between a pair of routes R_p and R_q; to do so we replace each subset of customers $S_1 \subseteq R_p$ of size $|S_1| \leq \lambda$ with any other one $S_2 \subseteq R_q$ of size $|S_2| \leq \lambda$. This way, we obtain two new routes $R'_p = (R_p - S_1) \bigcup S_2$ and $R'_q = (R_q - S_2) \bigcup S_1$, which are part of the new solution $S' = \{R_1 \ldots R'_p \ldots R'_q \ldots R_k\}$.

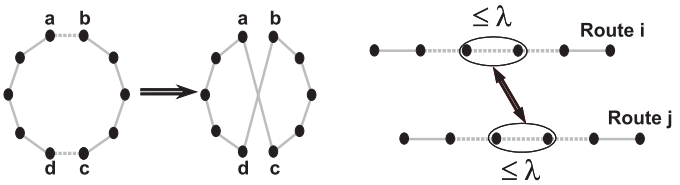


Fig. 13.5. 2-Opt works into a route (**a**), while λ-Interchange affects two routes (**b**)

Therefore, 2-Opt searches for better solutions by modifying the order of visiting customers inside a route, while the λ-Interchange method results in customers either being shifted from one route to another, or customers being exchanged between routes. This local search step is applied to an individual after the recombination and mutation operators, and returns the best solution between the best ones found by 2-Opt and 1-Interchange, or the current one if it is better (see a pseudo-code in Alg. 13.3). In the local search step, the algorithm applies 2-Opt to all the pairs of non-adjacent edges in every route and 1-Interchange to all the subsets of up to 1 customer between every pair of routes.

Algorithm 13.3 The local search step

```
1. proc Local_Search(ind)        // 'ind' is the individual to improve
2. for s ← 0 to MAX_STEPS do
3.    //First: 2_Opt. K is the number of routes. MAX_STEPS = 20
4.    best2_Opt = ind;
5.    for r ← 0 to K do
6.       sol = 2_Opt(ind,r);
7.       if Best(sol,best2_Opt) then
8.          best2_Opt = sol;
9.       end if
10.   end for
11. end for
12. //Second: 1_Interchange.
13. best1_Interchange = ind;
14. for s ← 0 to MAX_STEPS do
15.   for i ← 0 to Length(ind) do
16.      for j ← i+1 to Length(ind) do
17.         sol = 1_Interchange(i,j);
18.         if Best(sol,best1_Interchange) then
19.            best1_Interchange = sol;
20.         end if
21.      end for
22.   end for
23. end for
24. return Best(best2_Opt, best1_Interchange, ind);
25. end proc Local_Search;
```

In summary, the functioning of JCell is quite simple: in each generation, an offspring is obtained for every individual after applying the recombination operator (ERX) to its two parents (selected from its neighborhood). The offsprings are mutated with a special combined mutation, and later a local post-optimization step is applied to the mutated individuals. This local search algorithm consists in applying to the individual two different local search methods (2-Opt and 1-Interchange), and returns the best individual among the input individual and the output of 2-Opt and 1-Interchange. The population of the next generation will be composed of the current one after replacing its individuals with their offsprings in the cases where they are better.

13.3 Solving CVRP with JCell2o1i

In this section we describe the results of the experiments we have made for testing our algorithm on CVRP. JCell2o1i has been implemented in Java and tested on a Pentium IV 2.8 GHz PC with Linux on a very large test suite, composed of an extensive set of benchmarks drawn from the literature: (i) Augerat *et al.* (sets A, B and P) [29], (ii) Van Breedam [252], (iii) Christofides and Eilon [46], (iv) Christofides, Mingozzi and Toth (CMT) [47], (v) Fisher [87], (vi) Golden *et al.* [109], (vii) Taillard [240], and (viii) a benchmark generated from TSP instances [212]. All these benchmarks, as well as the best-known solution for their instances, are publicly available at [71].

Due to the heterogeneity of this huge test suite, we will find many different features in the studied instances, which will represent a hard test to JCell2o1i for this problem, usually much larger than usual studies in the VRP literature.

The parameterization used in this study for JCell2o1i (listed in Table 13.1) is the same for the 160 instances of our benchmark with the only exception of the maximum allowed number of evaluations of the termination condition. This value was fixed in terms of the length and difficulty of the instances. All the numerical results of the algorithm (optimal routes, costs, etc.) for every problem are available in [14]. These results were obtained after making 100 independent runs (in order to obtain statistical confidence), except for the case of the benchmark by Golden et al., for which 30 runs were made due to the high computational requirements of these problems.

Table 13.1. Parameterization used in JCell2o1i

Population size	100 individuals (10×10)
Parent selection	Current individual + binary tournament
Recombination	ERX, $p_c = 0.65$
Mutation of individuals	Insertion, Swap or Inversion (same prob.), $p_m = 0.85$
Bit mutation probability	$p_{bm} = 1.2/L$ (L = Individual length)
Replacement	Rep_if_not_Worse
Local search (LS)	2-Opt & 1-Interchange, 20 optimization steps

Table 13.2. Average deviation between our best solution found and the best known one for all the instances of every studied benchmark

Benchmark	Avg. Δ
Augerat et al. Set A	0.00
Augerat et al. Set B	$3.48e-03$
Augerat et al. Set P	0.00
Van Breedam	**2.45**
Christofides & Eilon	0.00
CMT	0.29
Fisher	0.00
Golden at al.	1.44
Taillard	0.56
Extended from TSP	0.00

We show in Table 13.2 the average deviations obtained between our best obtained solution (sol) and the best known one ($best$) for all the instances of every benchmark. This deviation (Δ) is computed as shown in Eq. 13.8.

$$\Delta(sol) = \left| \frac{best - sol}{best} \right| \cdot 100 \ . \tag{13.8}$$

Following the TSPLIB convention [212], distances between customers have been rounded to the closest integer value in instances belonging to the benchmarks of Augerat et al., Van Breedam, Christofides and Eilon, Fisher, and the set of translated instances from TSP.

As it can be seen in Table 13.2, the average errors obtained by JCell2o1i for every benchmark are very low, with values always under 1.5%. Note that the **bolded** result for Van Breedam's benchmark corresponds to an improvement of our results with respect to the previously existing ones. Thus, JCell2o1i has demonstrated to be a robust algorithm highly suitable for VRP on a large testbed.

13.4 New Solutions to CVRP

During the studies made on CVRP for this book, 10 new best-known solutions to CVRP were found. This supposes an important result for our work due to both the difficulty of the problem and the high number of researchers interested on it. These instances belong to three different benchmarks, two of them were studied in this chapter (Van Breedam and Taillard test suites), and the other one is the very large scale VRP (VLSVRP) benchmark studied in Chap. 8. We compile the new solutions found in Table 13.3. Note that the difference between the new best-known solution and the previously existing one is very little in some cases, but they represent different solutions. The most important differences were found for the instances of Van Breedam's benchmark, with an improvement of more than 5.5% in some case.

Table 13.3. New best solutions found

Inst.	New Best Solution	Δ (%)	Previous Best Solution
Bre-1	**1106.00**	3.24	1143.00 [253]
Bre-2	**1506.00**	4.68	1580.00 [253]
Bre-4	**1470.00**	0.41	1476.00 [253]
Bre-5	**950.00**	2.56	975.00 [253]
Bre-6	**969.00**	1.02	979.00 [253]
Bre-9	**1690.00**	5.59	1790.00 [253]
Bre-10	**1026.00**	1.82	1045.00 [253]
Bre-11	**1128.00**	2.76	1160.00 [253]
tai75b	**1344.62**	0.00	1344.64 [240]
vls26	**23977.73**	0.00	23977.74 [162]

13.5 Conclusions

In this chapter we have developed a single algorithm which is able to compete with the many different best known optimization techniques for solving the CVRP. Our algorithm has been tested in a very large set of 160 instances with different features, e.g., uniformly and not uniformly dispersed customers, clustered and not clustered, with a centered or not centered depot, having maximal route distances or not, considering drop times or not, with homogeneous or heterogeneous demands, etc.

We consider that the behavior of our cellular algorithm with merged mutation plus local search is very satisfactory since it obtains the best-known solution in most cases, or values very close to it, for all of the test suite. Moreover, it has been able to improve the known best solution so far for nine of the tested instances, which represents an important record in present research. Hence, we can say that the performance of JCell2o1i is similar or even better to that of the best algorithm for each instance. Besides, our algorithm is quite simple since we have designed a canonical cGA with three widely used mutations in the literature for this problem, plus two well-known local search methods.

Finally, we want to emphasize the good results we have reported in this book since, in addition to the nine new obtained solutions using JCell2o1i in this chapter, we must also consider the new best solution found in Chap. 8 for the VLSVRP problem. Moreover, in Chap. 8 we found the best solution (that was visually estimated and never found by any other algorithm) in six out of the twelve instances.

14

Telecommunications: Optimization of the Broadcasting Process in MANETs

I believe that if you show people the problems and you show them the solutions they will be moved to act.

Bill Gates (1955 -) – Businessman

In this chapter, we study the application of a multi-objective cGA for optimizing DFCN, a broadcasting protocol specially designed for metropolitan ad hoc networks (MANETs). Optimizing DFCN is a newly defined problem [18] consisting of finding the best values for a set of important parameters of the protocol that characterize its behavior. The objectives are to minimize the use of the network and the total process time, and to maximize on the other hand the coverage of the broadcasting protocol (the number of devices reached).

Mobile Ad Hoc Networks (MANETs) are fluctuating networks populated by a set of communicating *devices* (also called *nodes*) which can spontaneously interconnect each other without any pre-existing infrastructure. This means that no organization is present in such networks. The most popular wireless networking technologies available nowadays for building MANETs are Bluetooth and IEEE802.11 (WiFi). This implies that a) devices communicate within a limited range, and b) devices may move while communicating. A consequence of mobility is that the topology of such networks may change quickly and in unpredictable ways. This dynamical behavior constitutes one of the main obstacles for performing efficient communications.

In this chapter we are considering the problem of broadcasting on a particular subclass of MANETs called *Metropolitan* MANETs, which have some specific properties: the density in the network is heterogeneous and dynamic (particularly, high density regions do not remain active full time). The broadcasting strategy we are considering in this work is the so called *Delayed Flooding with Cumulative Neighborhood* protocol (DCFN) [132]. Three real world examples of such networks, a shopping mall, a metropolitan area, and a highway environment, have been taken into account so that, instead of providing a multi-purpose protocol, the originality of our proposal lies in tuning the broadcasting service for each particular network. Optimizing a broadcasting strategy implies multiple goals to be satisfied at the same time, such as maximizing the number of devices reached (coverage), minimizing the network use (bandwidth), and/or minimizing the duration of the process. Thus, what we

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_14, © Springer Science+Business Media, LLC 2008

are facing is known as a multi-objective optimization problem [65, 49], and we propose a new multi-objective cGA (cMOGA) for solving it. The results will be compared versus the main state-of-the-art algorithm in multi-objective optimization, NSGA-II [67].

As it was already mentioned in Chap. 9, the most popular algorithms for solving multi-objective problems are evolutionary algorithms [49, 65]. Though, very few works use GAs based on cellular models [146, 159, 190], even though cGAs have demonstrated to have very high efficiency and accuracy in mono-objective optimization [12, 17, 19, 99]. The proposed algorithm, cMOGA, represents a contribution to this field. Furthermore, this research line is, at the very best of our knowledge, the first attempt to solve the broadcasting problem on MANETs using a multi-objective EA.

The rest of this chapter is organized in the following manner. In Sect. 14.1 we describe the considered problem, the set of proposed scenarios for this problem, and the broadcasting strategy we plan to optimize. The proposed approach (a multi-objective cGA) is described in Sect 14.2. In Sect. 14.3 we present our experimentation and analyze the results. Our proposed algorithm is compared versus NSGA-II (the main state-of-the-art algorithm in the multi-objective domain) in Sect. 14.4. We finish this chapter with our main conclusions.

14.1 The Problem

The problem we study in this paper consists of, given an input metropolitan MANET network, determining the most adequate parameters for a broadcasting strategy. We first describe in Sect. 14.1.1 the target networks we use. Section 14.1.2 is devoted to the presentation of DFCN [132], the broadcasting strategy to be tuned. Finally, the MOPs we define for this work are presented in Sect. 14.1.3.

14.1.1 Metropolitan Mobile Ad Hoc Networks. The Madhoc Simulator

Metropolitan mobile ad hoc networks are MANETs with some particular properties. First, they have one or more areas where the node density is higher than the average. They are called *high density areas*, and they can be statistically detected. A high density area may be, for example, a supermarket, an airport, or an office. Second, high density areas do not remain active full time, i.e., they can appear and disappear from the network (e.g., a supermarket is open, roughly, from 9 a.m. to 10 p.m., and outside this period of time, the node density within the corresponding area is close to zero). An example of 4 km^2, 2000 devices metropolitan network is displayed in Fig. 14.1.

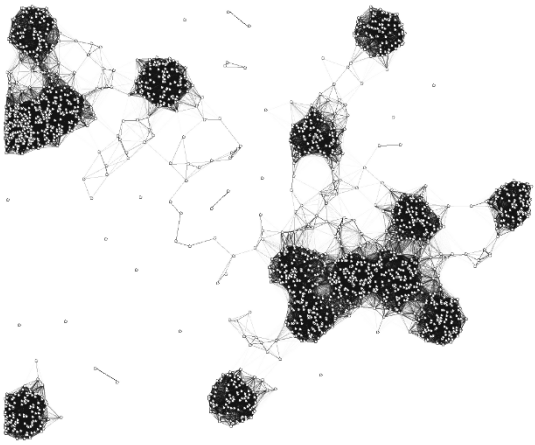


Fig. 14.1. An example metropolitan MANET

To deal with such kind of networks, there is no solution other than resorting to software simulations. In this work we have used *Madhoc*, a metropolitan MANET simulator [131][1]. It aims at providing a tool for simulating different level services based on distinct technologies on MANETs for different environments.

In the context of metropolitan MANETs, various topological configurations are very usually found. Some examples are networks built on-the-fly by people moving in concert places, market places, train stations, shopping centers, and city centers. All these scenarios have a number of different characteristics, such as the mobility and density of devices, the size of the area, and the presence or not of walls (which are obstacles for mobility and attenuate the signal strength), among others. Hence, three very different realistic scenarios, implemented by Madhoc, are used in this paper. These scenarios correspond to real world environments, and they aim at modelling a shopping mall, a metropolitan area, and a highway scenario.

- **Mall Environment.** The mall environment is proposed for emulating MANETs in commercial shopping centers, in which stores are usually located together one each other in corridors. People go from one store to another through these corridors, occasionally stopping for looking some shopwindows. These shopping centers are usually very crowded (high density of devices), and people behave differently (in terms of mobility) when they are in or out of the stores. Additionally, a high density of shops can be found in this kind of scenario. Finally, in the mall environment both the mobility of devices and their signal propagation are restricted by the walls of the building.

[1] The MANET simulator is freely available at `http://www-lih.univ-lehavre.fr/~hogie/madhoc/`

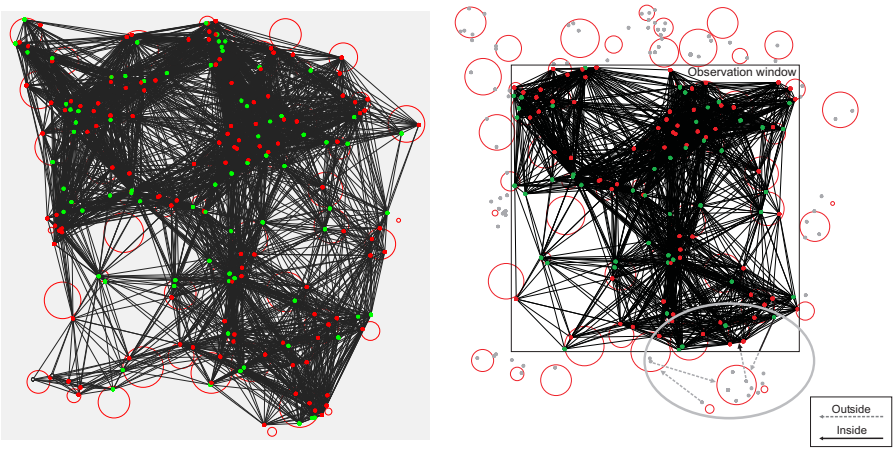


Fig. 14.2. The effects of introducing an observation window on the studied MANET

- **Metropolitan Environment.** The second realistic scenario we propose is the metropolitan environment. For that, we locate a set of spots (crossroads) in the modelled surface, and link them with streets. In this case, both pedestrians and vehicles are modelled, and they are continuously moving from one crossroad to another. Like in the real world, devices must reduce their speed when they approach a crossroad. In this scenario, the obstruction of the walls in the signal strength will be stronger than in the case of the mall environment.
- **Highway Environment.** We use this environment for simulating the behavior of MANETs out of cities. As an example, think on a large surface with roads, and people travelling by car. In this context, there should be a very low density of devices per square kilometer (all the devices are located on the roads), moving all of them in a fast manner. Additionally, there should not exist obstacles that attenuate the signal strength in communications.

In order to make our studies more realistic, an *observation window* has been included in the simulations, such that only the devices located into this window are taken into account for measurements. This makes possible the simulation of nodes exiting and joining the network by entering or leaving the observation window, respectively. Therefore, we are allowing the existence of a changing number of devices in the network, as it is the case in real MANETs. In all our tests in this work, this observation window covers the 70% of the whole area. As an example, in Fig. 14.2 we can see a MANET simulating a mall environment (left), and the observation window we study (right); supposing the 70% of the whole network. The circles represent the shops, while the points stand for the devices (those outside the observation window are grey colored, meaning that they are not considered for measurements).

14.1.2 Delayed Flooding with Cumulative Neighborhood

Williams and Camp [263] and, more recently, Stojmenovic and Wu [237] proposed two of the most frequently referenced analysis of broadcasting protocols. In their proposal, Stojmenovic and Wu [237] stated that protocols can be classified according to their algorithmic nature –determinism (no use of randomness), reliability (guarantee of full coverage)– or the information required by their execution (network information, "hello" messages content, broadcast messages content). Similarly, Wu and Lou [267] categorized protocols as *centralized* [204] and *localized* ones. Centralized protocols require a global or quasi-global knowledge of the network. They are hence not scalable. Localized protocols are those which require some knownledge of the network at only 1 or 2-hops, being the 1-hop neighborhood of a given device the set of devices directly seen by itself, while the 2-hops neighborhood is the 1-hop one plus the neighbors of its neighbors.

Using the classifications presented above, the delayed flooding with cumulative neighborhood protocol (DFCN) [132] is a deterministic algorithm. It does not consist of a new approach for calculating dominating sets. Instead it is a fully localized protocol which defines heuristics based on 1-hop information. This permits DFCN to achieve great scalability. The "hello" messages interchanged by the nodes do not carry any additional information. Only broadcast messages must embed the list of node's neighbors.

For being able to run the DFCN protocol, the following assumptions must be met:

- Like many other neighbor-knowledge-based broadcasting protocols (FWSP, SBA, etc.) [166, 205], DFCN requires the knowledge of 1-hop neighborhood. This is obtained by the use of "hello" packets at a lower network layer. The set of neighbors of the device s is named $N(s)$.
- Each message m carries –embedded in its header– the set of IDs of the 1-hop neighbors of its most recent sender.
- Each device maintains local information about all messages received. Each instance of this local information consists of the following items:
  - the ID of the message received;
  - the set of IDs of the devices that are known to have received the message;
  - the decision of whether the message should be forwarded or not.
- DFCN requires the use of a random delay before possibly re-emitting a broadcast message m. This delay, called random assessment delay (RAD), is intended to prevent collisions. More precisely, when a device s emits a message m, all the devices in $N(s)$ receive it at the same time. It is then likely that all of them forward m simultaneously, and this simultaneity entails network collisions. The RAD aims at randomly delaying the retransmission of m. As every device in $N(s)$ waits for the expiration of a different RAD before forwarding m, the risk of collisions is hugely reduced.

DFCN is an event driven algorithm which can be divided into three main parts: the two first ones deal with the handling of outcoming events, which are (i) new message reception and (ii) detection of a new neighbor. The third part (iii) consists of the decision making for emission as a follow-up of one of the two previous events. The behavior resulting from message reception is referred to as *reactive* behavior; when a new neighbor is discovered, the behavior is referred as *proactive* behavior.

Let s_1 and s_2 be two devices in the neighborhood of one another. When s_1 sends a packet to s_2, it attaches to the packet the set $N(s_1)$. At reception, s_2 hence knows that each device in $N(s_1)$ has received the packet. The set of devices which has *potentially* not yet received the packet is then $N(s_2) - N(s_1)$. If s_2 re-emits the packet, the *effective* number of devices newly reached is maximized by the heuristic function: $h(s_2,\ s_1) = |N(s_2) - N(s_1)|$.

In order to minimize the network use caused by a possible packet re-emission, a message is forwarded only if the number of newly reached devices is greater than a given threshold. This threshold is a function of the number of devices in the neighborhood (the local network density) of the recipient device s_2. It is written "threshold($|N(s)|$)". The decision made by s_2 to re-emit the packet received from s_1 is defined by the boolean function:

$$B(s_2, s_1) = \begin{cases} \text{true if } h(s_2, s_1) \geq \text{threshold}(|N(s_2)|) \\ \text{false otherwise} \end{cases}.$$

If the threshold is exceeded, the recipient device s_2 becomes an emitter in turn. The message is effectively sent when the random delay (defined by the RAD) expires. The threshold function, which allows DFCN to facilitate the message re-broadcasting when the connectivity is low, depends on the size of the neighborhood n, as given by:

$$\text{threshold}(n) = \begin{cases} 1 & \text{if } n \leq \text{safeDensity} \\ \text{minGain} * n & \text{otherwise} \end{cases},$$

where *safeDensity* is the maximum safe density below DFCN always rebroadcasts and *minGain* is a parameter of DFCN used for computing the minimum threshold for forwarding a message, i.e., the ratio between the number of neighbors which have not received the message and the total number of neighbors.

Each time a device s discovers a new neighbor, the RAD for all messages is set to zero and, therefore, the messages are immediately candidate to emission. If $N(s)$ is greater than a given threshold, which we have called *proD*, this behavior is disabled, so no action is undertaken on new neighbor discovery.

14.1.3 MOPs Definition

In the current section we define two new multi-objective problems for the optimization of the broadcasting protocol in MANETs. From the description of DFCN in the previous section, the following parameters are to be tuned:

minGain is the minimum gain for rebroadcasting. This is the most important parameter for tuning DFCN, since minimizing the bandwidth should be highly dependent on the network density. It ranges from 0.0 to 1.0.

[lowerBoundRAD,upperBoundRAD] define the RAD value (random delay for rebroadcasting in milliseconds). The two parameters take values in the interval $[0.0, 10.0]$ milliseconds.

proD is the maximal density ($proD \in [0, 100]$) for which it is still needed using proactive behavior (i.e., reacting on new neighbors) for complementing the reactive behavior.

safeDensity defines a maximum safe density of the threshold "threshold(n)" which ranges from 0 to 100 devices.

These five parameters, i.e., five decision variables which correspond to a DFCN configuration, characterize the search space of our MOP. We have set wide enough intervals for the values of these parameters in order to include all the reasonable possibilities we can find in a real scenario. Notice that some of the parameters are integer values, while some other ones are real numbers, so the optimization algorithm must deal in both the discrete and continuous domains at the same time. The objectives to optimize are (see Eq. 14.1): minimizing the duration of the broadcasting process, maximizing the network coverage, and minimizing the number of transmissions. Thus, we have defined a triple objective MOP, which is called DFCNT (which stands for DFCN Tuning). As we stated before, this problem is defined by a given target network in which the DFCN broadcasting strategy is used. Since three different real world metropolitan MANETs have been considered, three instances of DFCNT are to be solved: *DFCNT.Mall*, *DFCNT.Metropolitan*, and *DFCNT.Highway*.

$$DFCNT \begin{cases} f_1(x) = \text{broadcasting process length} & -\text{minimize} \\ f_2(x) = \text{coverage} & -\text{maximize} \\ f_3(x) = \text{number of transmissions} & -\text{minimize} \end{cases} \tag{14.1}$$

14.2 A Multi-objective cGA: cMOGA

In this section we present a multi-objective algorithm based on a canonical cGA model. As it was already mentioned in Chap. 9, although other cellular-like genetic approaches exist in the literature, to the best of our knowledge, none of them follows the canonical cGA model. In Alg. 14.1, we show the pseudo-code of cMOGA. We can observe that Alg. 14.1 is very similar to the pseudo-code of a canonical cGA (see Chap. 1). One of the main differences between the two algorithms is the existence of a *Pareto front* in the multi-objective case. The Pareto front is just an additional population composed of the non-dominated solutions found so far, which has a maximum size. In order to manage the insertion of solutions into the Pareto front with the goal of obtaining a diverse set, a *crowding* procedure has been used [67].

Algorithm 14.1 Pseudo-code of cMOGA

```
1. proc Evolve(cmoga)       //Parameters of the algorithm in 'cmoga'
2. Pareto_Front = NewFront() //Creates a new empty Pareto Front
3. while ! StopCondition() do
4.    for individual ← 1 to cmoga.popSize do
5.       neighbors ← GetNeighbors(cmoga,position(individual));
6.       parents ← Selection(neighbors);
7.       offspring ← Recombination(cmoga.Pc,parents);
8.       offspring ← Mutation(cmoga.Pm,offspring);
9.       Evaluation(offspring);
10.      Replacement(position(individual),aux_pop,offspring);
11.      InsertInParetoFront(individual);
12.   end for
13.   cmoga.pop ← aux_pop;
14. end while
15. end proc Evolve;
```

The cMOGA algorithm starts by creating an empty Pareto front (line 2 in Alg. 14.1). Individuals are arranged in a 2-dimensional toroidal grid, and the genetic operators are successively applied to them (lines 7 and 8) until the termination condition is met (line 3). Hence, for each individual, the algorithm consists of selecting two parents from its neighborhood, recombining them in order to obtain an offspring, mutating it, evaluating the resulting individual, and inserting it if it is not dominated by the current individual in both, the auxiliary population and the Pareto front (following a crowding procedure) —lines 10 to 13. Finally, after each generation, the old population is replaced by the auxiliary one.

14.2.1 Dealing with Constraints

To deal with constrained MOPs, cMOGA uses a simple approach also encountered in other multi-objective evolutionary algorithms, like NSGA-II [67] and microGA [48]. Whenever two individuals are compared, their constraints are checked. If both are feasible, a Pareto dominance test is directly applied. If one is feasible and the other is infeasible, the former dominates. In other case, if the two individuals are infeasible, then the one with the lowest amount of constraint violation dominates the other.

14.3 Experiments

In this section, we first describe the parameterization used by cMOGA. Next, the configurations of the network simulator for the three defined environments are described.

Table 14.1. Parameterization used in cMOGA

Population size	100 individuals (10×10)
Stop condition	25000 function evaluations
Neighborhood	NEWS
Parent selection	Current individual + binary tournament
Recombination	Simulated binary, $p_c = 1.0$
Mutation	Polynomial, $p_m = 1.0/L$ (L = Individual length)
Replacement	Rep_if_not_Dominates
Archive size	100 individuals
Crowding procedure	Adaptive grid

cMOGA has been implemented in Java inside the JCell framework and tested on a PC with a 2.8 GHz Pentium IV processor with 512 MB of RAM memory, and running SuSE Linux 8.1 (kernel 2.4.19-4GB). The Java version used is 1.5.0_05.

14.3.1 Parameterization of cMOGA

In Table 14.1 we show the parameters used by cMOGA. A square toroidal grid of 100 individuals has been chosen for structuring the population. The neighborhood used is composed of 5 individuals: the considered one plus those located at its North, East, West, and South (NEWS neighborhood). Due to the stochastic nature of the Madhoc simulator, five simulations per function evaluation are performed and the fitness values of the functions are computed as the average of the values obtained in each of these simulations. This important detail has a considerable influence in the running time to solve the problem, and explains why reporting 30 independent runs of the algorithm in our tests represents a high effort in studying this problem, since we want to ensure statistical confidence on the results.

We use the simulated binary recombination operator (SBX) [66] with probability $p_c = 1.0$. As its name suggests, SBX simulates the working principle of the single-point crossover on binary genotypes. The mutation operator used is the so called *polynomial* [66], and it is applied to every allele of the individuals with probability $p_m = 1.0/L$ (where L is the individual length). The resulting offspring replaces the individual at the current position if the latter does not dominate the former. For inserting the individuals in the Pareto front, an *adaptive grid algorithm* [151] is used. It consists of dividing up the objective space in hypercubes with the goal of balancing the density of non-dominated solutions in the hypercubes. Then, when inserting a non-dominated solution in the Pareto front, its grid location in the solution space is determined. If the Pareto front is already full and the grid location of the new solution does not match with the most crowded hypercube, a solution belonging to that most crowded hypercube is removed before inserting the new one.

Table 14.2. Main features of the proposed environments

		Mall	Metropolitan	Highway
Surface		40 000 m^2	160 000 m^2	1 000 000 m^2
Density of spots		800 (shops/km^2)	50 (crossroads/km^2)	3 (joints/km^2)
Devices	Speed out of spots	0.3–1 m/s	1–25 m/s	30–50 m/s
	Speed in spots	0.3–0.8 m/s	0.3–10 m/s	20–30 m/s
	Density	2 000 dev./km^2	500 dev./km^2	50 dev./km^2
Wall obstruction		70%	90%	0%

14.3.2 Madhoc Configuration

As we stated in Sect. 14.1.1, we have defined three different environments for MANETs modelling three possible scenarios that can be found in real world. Their main features are explained in this section, and they are summarized in Table 14.2. We show in Fig. 14.3 example MANETs for each of the studied scenarios (grey point are not considered because they are out of the observation window). These example networks were obtained by using the graphical interface of Madhoc with exactly the same parameterization suggested in our proposed benchmark. We consider that the broadcasting is completed when either the coverage reaches 100% or it does not vary in a reasonable period of time (set to 1.5 seconds after some preliminary experimentation). This point is important since an improper termination condition will lead us to bad results or slow simulations.

The Mall Environment

In this section we proceed to explain the parameterization we used for modelling the mall environment. In malls, the density of both shops (spots) and devices is usually very high. Additionally, there exist walls which attenuate the signal and limit the movements of devices, and these movements are usually very slow, since we are modelling people walking. We have defined for this work a shopping center of 200×200 square meters of surface with densities of 800 stores and 2 000 devices per square kilometer. Stores are circles of radius between 1 and 10 meters, and the obstruction of the walls is computed with a penalty of the 70% in the signal strength. Finally, devices travel with a speed ranging between 0.3 and 1 m/s in the corridors and between 0.3 and 0.8 m/s when they are inside the stores.

Regarding the mall environment, it is worth noting that the resulting connection graph is quite dense (see Fig. 14.3). The reason for that is that the coverage of mobile devices ranges between 40 and 80 meters (randomly selected value), and the simulation area is small. Hence, *DFCNT.Mall* is more difficult to solve due to the broadcast storm problem [198]. This problem consists of severe network congestions provoked by packet re-emissions due to frequent packet collisions.

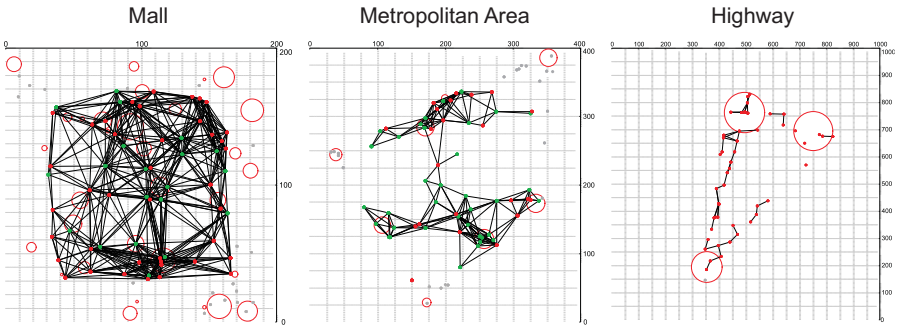


Fig. 14.3. The three studied scenarios for MANETs

The Metropolitan Environment

In this second environment we study the behavior of DFCN in a metropolitan MANET. For modelling this environment we set a surface of 400×400 square meters, with a density of 50 spots (standing for crossroads) per square kilometer having a circle surface of radius between 3 and 15 meters. The wall obstruction is in this case stronger than for the mall environment (up to 90%), and the density of devices is 500 elements per square kilometer. When setting the speed of devices, the cases when people move on foot or by car must be taken into account, so its value ranges between 0.3 and 10 m/s when they are in a crossroad, and between 1 and 25 m/s in other case (streets).

In Fig. 14.3, it can be seen that the resulting network in a metropolitan area is not as dense as that of the mall environment. Generally speaking, this kind of network is composed of a few number of subnetworks, which are usually connected one each other by only a few links, typically one or two, or even zero (unconnected subnetworks). Additionally, some devices could not be part of any subnetwork (isolated nodes). The topology of this network can change in a fast manner, since some of the devices are travelling in vehicles at high speeds. All these features difficult the broadcasting task, and that makes interesting the study of this kind of networks for us.

The Highway Environment

As we previously commented in Sect. 14.1.1, this environment is composed by a few number of devices, travelling at high speeds. This network has the peculiarity of having the wall obstruction set to 0%. The simulated surface was set to 1000×1000 square meters with a density of only 50 devices per square kilometer. These devices travel at random speeds between 30 and 50 m/s. We define the roads as the straight line connecting two spots, and we establish a density of only 3 spots (highway entrances and/or exits) for modelling the scenario. The speed of devices in the spots must be reduced to the range between 20 and 30 m/s. The size of each spot (length of the entrance/exit) is set to a random value between 50 and 200 meters (spots radius $\in [25, 100]$ meters).

Table 14.3. Results of cMOGA for the three instances *DFCNT*

Environment	Time (h)	Number of Pareto optima
DFCNT.Mall	66.12 ± 7.94	96.77± 3.20
DFCNT.Metropolitan	108.21± 8.41	95.59± 4.58
DFCNT.Highway	47.57 ± 0.42	52.23±11.04

It can be seen in Fig. 14.3 how the resulting network using this parameterization is composed by a set of multiple (usually disconnected) sub-networks involving a small number of devices (even only one). The existence of these small and unconnected networks supposes a hard obstacle for the task of the broadcasting protocol. Additionally, the topology of the network changes very fast due to the high speed on the devices movement. Hence, as a consequence of these high speeds, new subnetworks are continuously being made and disappearing, what hinders the broadcasting process even more.

14.3.3 Results for DFCNT

We now turn to present and analyze the obtained results for the *DFCNT* problem (Sect. 14.1.3) with the three environments. Let us remind that this problem is composed of five decision variables and three objective functions. All the values presented are the average over 30 independent runs of cMOGA.

In Table 14.3 we show the mean and the standard deviation of the execution time (in hours) and the number of non-dominated solutions found by cMOGA for the three *DFCNT* instances: *DFCNT.Mall*, *DFCNT.Metropolitan* and *DFCNT.Highway*. As can be seen, the execution time of a single run is very long, since it ranges from 1.98 days for the highway scenario, up to 4.51 days in the case of *DFCNT.Metropolitan*. The reason is the high cost of computing the fitness function, since we launch five simulations per evaluation, and each run of the simulator requires between 1 and 4 seconds. Regarding the number of solutions found, the obtained results are highly satisfactory in the three *DFCNT* instances, since the number of different solutions found is 96.77, 95.59, and 52.23 on average (the maximum is 100) for *DFCNT.Mall*, *DFCNT.Metropolitan* and *DFCNT.Highway*, respectively. Thus, we allow the decision maker to choose from a wide range of possibilities. Notice that the number of solutions composing the Pareto front decreases when decreasing the device density of the network (i.e., increasing the surface and decreasing the number of devices). This is because we stick to just one single criterion for considering that the broadcasting process is done in three very different environments. As a future work, we plan to customize this criterion for each environment, what hopefully will lead us to obtain still better results.

As an example of the diverse and wide set of solutions reported by the multi-objective optimizer, we plot in Fig. 14.4 an example Pareto front obtained with cMOGA for the three studied environments. Best solutions are those implying (i) high coverage, (ii) low bandwidth, and (iii) a short dura-

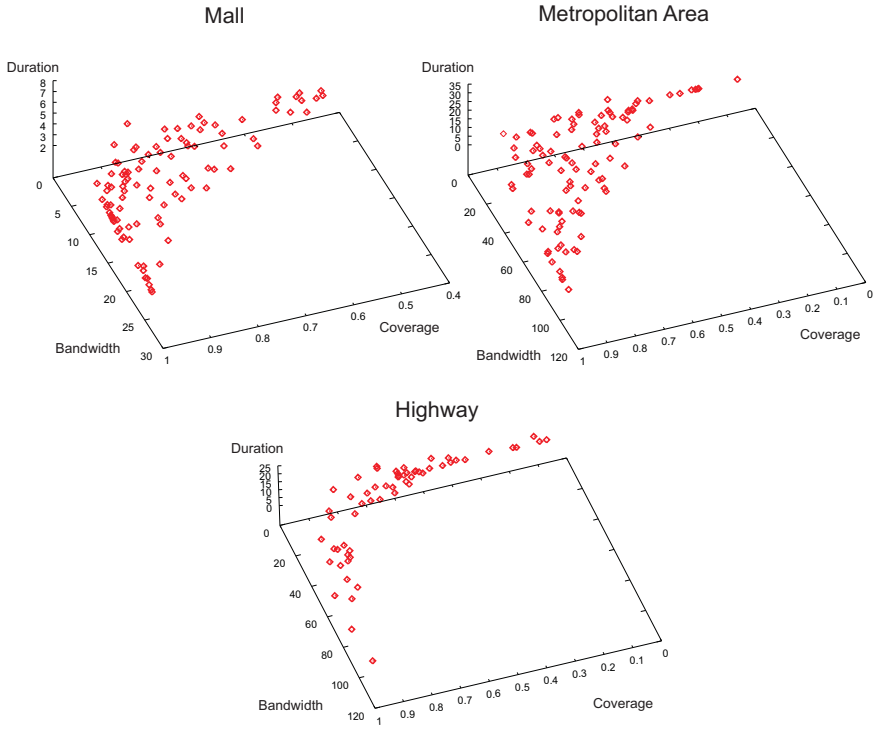


Fig. 14.4. Example Pareto Fronts for the three environments

tion of the broadcasting process, (i) and (ii) being the most important parameters. In fact, Pareto optima in these fronts reaching a coverage over 95% need on average 3.77 seconds and 17.25 messages for mall, and 13.78 seconds and 75.71 messages (i.e., intense bandwidth usage) in the case of the metropolitan area. In contrast, for the highway environment only 5 solutions of the Pareto front have a coverage over 95% (38 for *DFCNT.Mall* and 16 for *DFCNT.Metropolitan*), achieving an average of 74.88 messages sent in 13.37 seconds, which are values similar to those of the metropolitan area. Finally, notice that if coverage were not a hard constraint in our network, we could use very cheap solutions in terms of time and bandwidth for the two environments.

Comparing the three graphics, it is possible to observe that the broadcasting is more efficient in the mall environment than in the other two cases, since it takes less than 8 seconds (duration), transmitting always less than 23 messages (bandwidth), and reaching more than 40% of the devices (coverage). However, in the metropolitan and highway environments the broadcasting process is usually longer, with a larger number of transmitted messages, and the coverage is in some cases less than 10%. Finally, although the front obtained for the highway environment has similar bounding values as those observed in the case of the metropolitan area, the diversity is much lower in the highway scenario.

The difference on the coverage of solutions found in the three environments is a common sense result, since the probability of having isolated subnetworks (composed of one or more devices) grows when increasing the simulation surface and decreasing the number of devices. Hence, the difference in the quality of the solutions is a consequence of the different topology of the networks, since the high connectivity of the devices in the mall environment allows one message to reach many more devices than in the case of the other two studied environments. Conversely, this high connectivity increases the risk of a broadcast storm, making *DFCNT.Mall* very difficult to solve. From our results we can conclude that cMOGA has dealt with the problem successfully.

The Pareto fronts on Fig. 14.4 fulfill the design objectives of the DFCN protocol: most of the plots (in the center of the clouds) provide sets of parameters which make DFCN achieving a coverage rate close to 100%, keeping the network throughput very low. What makes the DFCNT problem particularly interesting from an applicative point of view is that it permits the decision maker to discard this default behavior by setting a *degree of coverage* for the broadcasting application. Indeed not all applications require the maximization of the coverage rate. For example, *local advertising* –which consists in spreading advertisement messages to devices a few hops away from the source– needs the broadcasting process to cease after a while. Sometimes high coverage is even to be avoided. For example trying to achieve a high coverage on metropolitan MANETs (which may realistically be made of thousands devices) is harmful, since it is likely to lead to severe network congestions.

14.4 Comparing cMOGA Against NSGA-II

In this section we make a comparative study of the behavior of cMOGA and NSGA-II, a state-of-the-art metaheuristic in multi-objective optimization, on DFCNT problem. For this comparison, the three studied environments (mall, metropolitan and highway) were solved with NSGA-II.

14.4.1 Parameterization of NSGA-II

We present in this section the parameterization used for NSGA-II. This algorithm stops when 25000 function evaluations are made, and has a maximuma size of 100 non-dominated solutions for the archive. The recombination and mutation operators used are SBX and polynomial, respectively, and they two use a distribution index of 10 units. The recombination probability is set to 0.9, while in the case of mutation the probability is 0.2 (as it was suggested by the designers of the algorithm). As in the case of cMOGA, five independent runs of the simulator are run for evaluating the individuals. It is important to note that 25000 function evaluations $\times$ 5 simulations/evaluation means that DFCN has been optimized in more than 125000 different network instances.

Table 14.4. NSGA-II versus cMOGA for DFCNT. Number of Pareto optima

		$\overline{x}$	max	min	Test
DFCNT.Mall	NSGA-II	$99.24_{\pm 1.5885}$	100	93	+
	cMOGA	$96.77_{\pm 3.2022}$	99	87	
DFCNT.Metropolitan	NSGA-II	$83.52_{\pm 15.2728}$	100	41	+
	cMOGA	$95.59_{\pm 4.5788}$	99	80	
DFCNT.Highway	NSGA-II	$54.92_{\pm 14.0083}$	81	24	−
	cMOGA	$52.23_{\pm 11.0429}$	76	34	

Table 14.5. NSGA-II versus cMOGA for DFCNT. Hypervolume metric

		$\overline{x}$	max	min	Test
DFCNT.Mall	NSGA-II	$0.6220_{\pm 0.0115}$	0.6590	0.6125	+
	cMOGA	$0.8337_{\pm 0.0917}$	0.9919	0.7041	
DFCNT.Metropolitan	NSGA-II	$0.7648_{\pm 0.0152}$	0.7860	0.7412	+
	cMOGA	$0.9261_{\pm 0.0461}$	1.0000	0.8319	
DFCNT.Highway	NSGA-II	$0.8859_{\pm 0.0122}$	0.9234	0.8660	−
	cMOGA	$0.9391_{\pm 0.1371}$	1.0000	0.3274	

Table 14.6. NSGA-II versus cMOGA for DFCNT. Set coverage metric

	$\mathcal{C}(\mathbf{A},\mathbf{B})$					
	A	**B**	$\overline{x}$	max	min	Test
DFCNT.Mall	NSGA-II	cMOGA	$0.9467_{\pm 0.0231}$	0.9894	0.8854	+
	cMOGA	NSGA-II	$0.0348_{\pm 0.0464}$	0.1837	0.0000	
DFCNT.Metropolitan	NSGA-II	cMOGA	$0.8798_{\pm 0.0343}$	0.9457	0.8333	+
	cMOGA	NSGA-II	$0.6319_{\pm 0.1045}$	0.8267	0.4143	
DFCNT.Highway	NSGA-II	cMOGA	$0.7665_{\pm 0.0787}$	1.0000	0.6053	−
	cMOGA	NSGA-II	$0.8256_{\pm 0.2204}$	1.0000	0.0000	

14.4.2 Discussion

The comparison of the two algorithms (cMOGA and NSGA-II) is made in this section. The results are obtained after making 30 independent runs of every experiment, and metrics *hypervolume* and *set coverage*, as well as the number of non-dominated solutions found in the Pareto front were used for comparing the algorithms (see Chap. 9).

The obtained results are displayed in Tables 14.4 to 14.6, and they include the average, $\overline{x}$, and the standard deviation of the obtained results. The maximum and minimum obtained values (max and min, respectively) are also provided for each metric. The same statistical study discussed in Chap. 5 has been applied here in order to obtain statistical relevance in our conclusions.

As it can be seen in Table 14.4, cMOGA finds a higher number of non-dominated solutions in the case of the metropolitan environment, while it is worse than NSGA-II for the mall scenario. In the case of *DFCNT.Highway* problem no statistically significant differences were found. Regarding the hy-

pervolume metric (see Table 14.5), cMOGA outperforms NSGA-II for the three studied problems, with statistical significance for *DFCNT.Mall* and *DFCNT.Metropolitan* problems. Finally, in contrast to the observed results for the two previous metrics, NSGA-II is better than cMOGA in terms of the set coverage metric (as can be seen in Table 14.6). Our algorithm only improves NSGA-II in the Highway environment, but the difference is not significative in this case.

Thus, summarizing the obtained results, there is no algorithm clearly better than the other one. On the one hand, cMOGA seems to be better than NSGA-II in terms of the number of non-dominated solutions found and the hypervolume metric. On the other hand, it is outperformed by NSGA-II in the case of the set coverage metric. The differences between the two algorithms are always significative for *DFCNT.Mall* and *DFCNT.Metropolitan*, while we did not find important differences in the case of *DFCNT.Highway*.

14.5 Conclusions

This chapter presents a first approximation to the problem of optimally tuning DFCN, a broadcasting protocol for MANETs, using cMOGA, a new cellular multi-objective genetic algorithm. cMOGA has been used to solve *DFCNT*, which is defined as a three-objective MOP, with the goals of minimizing the duration of the broadcasting process, maximizing the network coverage, and minimizing the network usage.

Three different realistic scenarios were used. We have then solved three different instances of the new presented MOP. They correspond to a shopping center (*DFCNT.Mall*), the streets in a city (*DFCNT.Metropolitan*), and a wide non-metropolitan area wherein several roads exist (*DFCNT.Highway*). Our experiments reveal that solving *DFCNT* instances provides a populated Pareto front composed of more than 50 points for *DFCNT.Highway*, and more than 95 points (i.e., more than 95 different DFCN configurations) in the case of the other two environments, all this at the expense of a considerable amount of time. However, this time can be affordable for a network designer.

In the second part of this chapter we compared our new algorithm (cMOGA) versus NSGA-II, a multi-objective algorithm that belongs to the state of the art, for the three proposed problems. Three different metrics were used for comparing the algorithms: the number of Pareto optima, the set coverage and hypervolume metrics. From the results obtained we can not conclude that any of the two algorithms is better than the other. We observed that cMOGA outperforms NSGA-II in terms of the hypervolume metric, while NSGA-II is better than cMOGA for the set coverage metric. Regarding the number of non-dominated solutions found, there is no a winner algorithm, since both of them outperform the other one for one of the three studied problem (there is one problem for which no significant differences were found).

15

Bioinformatics: The DNA Fragment Assembly Problem

To be prepared is half the victory.

Miguel de Cervantes (1547 - 1616) – Spanish Novelist

In previous chapters, cellular GAs were applied to different domains such as telecommunications, logistics, and mathematics in order to demonstrate we are prepared for solving complex real-world problems when equipped with these techniques. In this chapter we plan to complete this study by facing a problem taken from the bioinformatics field. Specifically, we consider in this chapter the DNA fragment assembly problem, which is an important problem that emerges in laboratories involved in genomic projects. The goal of this problem is to determine the complete sequence of the genome and its genetic contents. The genomic project is accomplished in two steps: the genome sequencing (we focus here on this first one), and the genome annotation (i.e., the process of identifying the boundaries between genes and other features in raw DNA sequences).

The reason for making these two steps in this process is the large strands of DNA that are needed to read nowadays (e.g., human DNA is about 3.2 billion nucleotides in length). Thus, these strands are broken into small fragments for sequencing in a process called *shotgun sequencing*. In this approach, several copies of a portion of DNA are each broken into many segments short enough to be sequenced automatically by a machine, but this process does not keep neither the ordering of the fragments nor the portion from which a particular fragment came. This leads to the DNA fragment assembly problem [228] in which these short sequences have to be reassembled to their (supposed) original form. The automation allows shotgun sequencing to proceed far faster than traditional methods. But comparing all the tiny pieces and matching up the overlaps requires massive computation.

The importance of accurately solving the DNA fragment assembling problem is very high since it must be faced in the early phases of the genome project, and thus the other steps depend on the quality of the obtained results. This problem is a very difficult combinatorial optimization problem (it is NP-hard), and it is growing in importance and complexity as more research centers become involved on sequencing new genomes. The difficulty of the problem is given by its huge search space even in the absence of noise, since given k fragments, there are $2^k k!$ possible combinations.

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*,
doi: 10.1007/978-0-387-77610-1_15, © Springer Science+Business Media, LLC 2008

The DNA fragment assembly problem has been already tackled with various different heuristics in the literature (including genetic algorithms), but due to its importance and complexity, more accurate and (specially) faster tools must be developed. Recently, Alba and Luque have proposed in [22] a new heuristic algorithm which is more accurate and much faster than all the previously existing ones for this problem. The name of this new heuristic is PALS (which stands for problem aware local search), and it is used in this chapter for hybridizing a cGA in order to improve the accuracy of the algorithm. This should be a very successful result due to the importance of this problem on the genome project, even though the execution speed is penalized (with respect to PALS itself).

This chapter is organized in the following manner. The problem of the DNA fragment assembly is described in Sect. 15.1, and the new hybrid cGA we propose for solving it is presented in Sect. 15.2. Finally, our main results are shown and discussed in Sect. 15.3, and our conclusions are given in Sect. 15.4.

15.1 The DNA Fragment Assembly Problem

In order to determine the function of specific genes, scientists have learned to read the sequence of nucleotides comprising a DNA sequence in a process called DNA sequencing. To do that, multiple exact copies of the original DNA sequence are made. Each copy is then cut into short fragments at random positions. These are the first three steps depicted in Fig. 15.1 and they take place in the laboratory. After the fragment set is obtained, a traditional assemble approach is followed in this order: overlap, layout, and then consensus. To ensure that enough fragments overlap, the reading of fragments continues until a coverage is satisfied. These steps are the last three ones in Fig. 15.1. In what follows, we give a brief description of each of the three phases, namely overlap, layout, and consensus.

Overlap Phase – Finding the overlapping fragments. This phase consists of finding the best or longest match between the suffix of one sequence and the prefix of another. In this step, we compare all possible pairs of fragments to determine their similarity. Usually, a dynamic programming algorithm applied to semiglobal alignment is used in this step. The intuition behind finding the pairwise overlap is that fragments with a significant overlap score are very likely next to each other in the target sequence.

Layout Phase – Finding the order of fragments based on the computed similarity score. This is the most difficult step because it is hard to tell the true overlap due to the following challenges:

1. Unknown orientation: After the original sequence is cut into many fragments, the orientation is lost. One does not know which strand should be selected. If one fragment does not have any overlap with another, it is still possible that its reverse complement might have such an overlap.

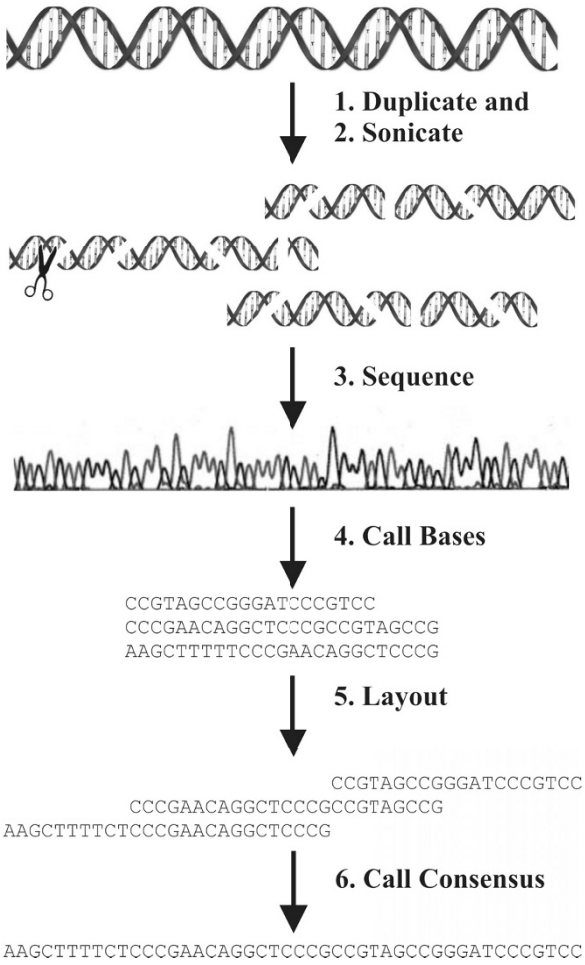


Fig. 15.1. Graphical representation of DNA sequencing and assembling

2. Base call errors: There are three types of base call errors: substitution, insertion, and deletion errors. They occur due to experimental errors in the electrophoresis procedure (the method used in the laboratories to read the DNA sequences). Errors affect the detection of fragment overlaps. Hence, the consensus determination requires multiple alignments in highly coverage regions.
3. Incomplete coverage: It happens when the algorithm is not able to assemble a given set of fragments into a single contig. A contig is a sequence in which the overlap between adjacent fragments is greater than or equal to a predefined threshold (cutoff parameter).

4. Repeated regions: Repeats are sequences that appear two or more times in the target DNA. Repeated regions have caused problems in many genome sequencing projects, and none of the current assembly programs can handle them perfectly.
5. Chimeras and contamination: Chimeras arise when two fragments that are not adjacent or overlapping on the target molecule join together into one fragment. Contamination occurs due to the incomplete purification of the fragment from the vector DNA.

After the order is determined, the progressive alignment algorithm is applied to combine all the pairwise alignments obtained in the overlap phase.

Consensus Phase – Deriving the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule in building the consensus.

To measure the quality of a consensus, we can look at the distribution of the coverage. Coverage at a base position is defined as the number of fragments at that position. It is a measure of the redundancy of the fragment data, and it denotes the number of fragments, on average, in which a given nucleotide in the target DNA is expected to appear. It is computed as the number of bases read from fragments over the length of the target DNA [228].

$$\text{Coverage} = \frac{\sum_{i=1}^{n} \text{length of fragment } i}{\text{target sequence length}} \ , \tag{15.1}$$

where n is the number of fragments. The higher the coverage, the fewer number of graphs, and the better the result.

15.2 A cMA for DNA Fragment Assembly Problem

In this section, a new cellular memetic algorithm (called cGA+PALS) for solving the DNA fragment assembly problem is presented. A pseudo-code of the cMA template we use in this chapter is shown in Alg. 15.1, and the parameterization used in this study is given in Table 15.1.

Solutions are represented as a permutation of integer numbers, and the value of each gene corresponds to one of the available fragments. The order of the numbers in the permutation represents the order of the fragments in the final solution. The fitness function used is the sum of the overlapping degrees of all the fragments.

A population of 400 individuals (arranged in a 20×20 grid) is used in our algorithm, and a NEWS neighborhood structure is considered. For the breeding loop, one of the parents is the current individual itself, while the other one is selected by binary tournament within its neighborhood. The two parents are then recombined using the order based crossover (OX). This operator lies in randomly selecting two cut points, and copying the largest portion of the best parent into the offspring. The other genes of this offspring are copied from

Algorithm 15.1 Pseudo-code of cMA+PALS

```
1. proc Evolve(cma)        //Algorithm parameters in 'cma'
2. GenerateInitialPopulation(cma.pop);
3. Evaluation(cma.pop);
4. while !StopCondition() do
5.    for individual ← 1 to cma.popSize do
6.       neighbors ← GetNeighbors(cma,position(individual));
7.       parents ← Select(neighbors);
8.       offspring ← Recombination(cma.Pc,parents);
9.       offspring ← Mutation(cma.Pm,offspring);
10.      offspring ← PALS(cma.Pl,offspring);
11.      Evaluation(offspring);
12.      InsertIfNotWorse(position(x,y),offspring,cma,aux_pop);
13.   end for
14.   cma.pop ← aux_pop;
15.   UpdateStatistics(cma);
16. end while
17. end proc Evolve;
```

the worst parent but not repeating any gene value and preserving their order of appearance in this worst parent. The resulting offspring is then mutated applying the swap operator to all the genes with equal probability. The swap operator is to exchange the value of a given gene with that of other randomly selected gene.

The local search step is an invocation to the PALS heuristic [22] with a maximum of 100 iterations (or until the current solution could not be improved in the last iteration of the algorithm), and it is applied to all the individuals in the population with the probability 1/PopSize. This parameterization was chosen after the results in previous studies that concluded that it is more efficient to apply an intensive local search to a very reduced number of individuals in every generation than applying a slighter step to all the individuals (see Chap. 7).

Table 15.1. Parameterization of cGA+PALS for the DNA fragment assembly problem

Population	20×20 individuals
Neighborhood	NEWS (Von Neumann)
Selection	Current individual + binary tournament
Recombination	OX, $P_c = 1.0$
Mutation	Swap, $P_m = 1/L$ (L = Individual length)
Local search	PALS (maximum = 100 steps), $P_{LS} = 1.0$/PopSize
Replacement	Rep_if_not_Worse
Stop condition	Reaching 60,000 evaluations

The PALS heuristic is a very recent technique much more efficient than any other previously existing algorithm for the DNA fragment assembly problem. One of the key issues of PALS is that contigs are taken into account when computing the quality of a solution. Since computing the contigs of a solution is a costly process, PALS calculates it by estimating the number of contigs that are created or destroyed when tentative solutions are manipulated. In contrast, most algorithms in the literature either do not take contigs into account (what would lead to the undesirable situation in which a solution can be better than another one with a larger number of contigs) or compute the number of contigs of the newly generated solutions by applying a final step of refinement with a greedy heuristic [163].

Finally, newly generated individuals replace the existing ones in the population if their quality (fitness value) is better or equal to that of the current individuals in the population. The stop condition of cGA+PALS is to reach 60,000 evaluations.

15.3 Results

We thoroughly compare in this section the results obtained by cGA+PALS, a cGA and the PALS heuristic itself. Additionally, these algorithms are compared against some of the most popular assemblers in the literature in terms of the final number of contigs assembled in the last part of this section.

In order to fairly compare the three studied algorithms, we made 100 independent runs and performed statistical tests on the obtained results. These algorithms were implemented in Java (using the JCell framework) and run on a Linux system with a Pentium IV 2.8 GHz processor and 512 MB of RAM. The parameters of cGA+PALS are those given in Sect. 15.2, and the cGA uses exactly the same parameters than cGA+PALS, but it does not perform any local search (it does not implement PALS). The parameterization used in the case of PALS is the one proposed in its original paper [22].

Thus, we have selected for our studies the two largest instances studied in [22], taken from the NCBI web site[1], corresponding to the Neurospora crassa (common bread mold) BAC, with accession number BX842596, which is 77,292 bases long. Some information about these specific fragment sets is given in Table 15.2. These instances are larger and more complex instances that researchers usually face (which are about 15-30k bases long).

The two studied instances are very hard since they are generated from very long sequences using a small/medium value of coverage and a very restrictive cutoff (threshold to join adjacent fragments in the same contig). The combination of these two parameters produces very complex instances.

We show in Tables 15.3 and 15.4 the results obtained by the studied algorithms for instance BX842596 with coverage 4 and 7, respectively. Specifically, we compare the algorithms in terms of the best and average solutions

[1] `http://www.ncbi.nlm.nih.gov`

Table 15.2. Information of datasets. The accession number is used as instance name

Parameters	**BX842596**	
Coverage	4	7
Mean fragment length	708	703
Number of fragments	442	773

Table 15.3. Comparison of the studied algorithms. Instance BX842596(4)

	Best Solution	**Average Solution**	**Best Contigs**	**Average Contigs**	**Time (seconds)**
PALS	227130	226022.49 $\pm$ 625.91	**1**	1.16 $\pm$0.44	**20.91** $\pm$ **0.21**
cGA	190118	186536.05 $\pm$1746.26	9	18.56 $\pm$3.83	159.59 $\pm$ 0.99
cGA+PALS	**227630**	**226923.42** $\pm$ **351.84**	**1**	**1.06** $\pm$**0.31**	280.99 $\pm$23.23
Test	•	+	•	+	+

found, the best and average number of contigs in the solution, and the average elapsed time. Their standard deviations are also included, and the best results are **bolded**. All the obtained results are statistically significant, except the case of the average number of contigs obtained by PALS and cGA+PALS for instance BX842596(4), for which we cannot statistically assure that are different with a 95% confidence level. Their standard deviations are also included.

It can be seen in these two tables that the proposed cMA is the best of the three compared algorithms for the two studied instances in terms of best and average solution found and number of contigs in the solution. Conversely, it is much slower than PALS, as it was expected. The regular cGA is the worst algorithm with statistically significant results in all the cases.

Hence, even thought the cGA is the worst compared algorithm, it becomes the best one after hybridizing it with the PALS heuristic as a local search step. However, although the run time is penalized, it is still highly competitive with the existing assemblers in the literature (which require several hours), and the quality of the solutions found by cGA+PALS is highly improved with respect to PALS. Indeed, cGA+PALS is the most accurate of the compared algorithms, since it obtains the highest average solutions with the lowest standard deviation values. Additionally, it is easy to considerably reduce the run time of the algorithm by implementing it in a more efficient language, such as C++.

The three studied algorithms are compared versus some of the most important assemblers existing in the literature in Table 15.5. These algorithms are a pattern matching algorithm (PMA) [163], and two commercial packages: CAP3 [135], and Phrap [123]. From this table we can conclude that the proposed cMA is a really competitive tool, clearly outperforming all the compared algorithms, and thus representing the new state of the art.

Table 15.4. Comparison of the studied algorithms. Instance BX842596(7)

	Best Solution	**Average Solution**	**Best Contigs**	**Average Contigs**	**Time (seconds)**
PALS	352718	343888.39 $\pm$3074.04	11	18.48 $\pm$ 4.00	**66.99** $\pm$ **0.12**
cGA	304714	296891.41 $\pm$3614.00	35	50.80 $\pm$ 6.43	255.11 $\pm$ 2.17
cGA+PALS	**445395**	**445134.95** $\pm$ **209.59**	**1**	**1.00** $\pm$**0.00**	2334.30 $\pm$164.69
Test	•	+	•	+	+

Table 15.5. Best final number of contigs for the studied assemblers and for other specialized systems

	PALS	**cGA**	**cGA+PALS**	**PMA [163]**	**CAP3 [135]**	**Pharp [123]**
BX842596(4)	**1**	9	**1**	7	6	6
BX842596(7)	11	35	**1**	2	2	2

15.4 Conclusions

We proposed in this chapter a new cellular memetic algorithm for solving the DNA fragment assembly problem. The algorithm was obtained after hybridizing a cGA with the PALS heuristic, a recently proposed heuristic that represents the state of the art for this problem. The resulting algorithm clearly improves the results of PALS, as well as those of an equivalent cGA without local search. However, the resulting cMA (called cGA+PALS) considerably increases the computational time with respect to PALS, although it is still faster than the compared algorithms we found in the literature. Typically, algorithms in the literature usually require several hours for solving the problem rather than the 38 minutes that the cMA lasts (in average) for the most computationally expensive instance of the two compared ones.

Part V

Appendix

A

Definition of the Benchmark Problems

We build too many walls and not enough bridges.

Isaac Newton (1642 - 1727) – Scientist

In order to make this book self-contained, we present in this appendix a brief description of all the problems used in our studies. In this way, we want to build a bridge between any chapter and this one, allowing the reader to access the details on the problems but not interrupting the smooth reading of the chapters with these details. We structure the description of all the problems in different sections according to which optimization field they belong to: combinatorial optimization in Sect. A.1, continuous optimization in Sect. A.2, and multi-objective optimization in Sect. A.3.

A.1 Combinatorial Optimization Problems

In this section we describe the different combinatorial optimization problems we have tackled in this book. In the definition of some of them the Hamming distance is used. The Hamming Distance between a bit string a and another b is measured by the equation:

$$d_{ab} = \sum_{i=1}^{l} a_i \otimes b_i \ . \tag{A.1}$$

Hence, identical bit strings have a hamming distance of $d_{ab} = 0.0$, whereas completely different bit strings will have a distance of $d_{ab} = l$, being l the length of the two strings.

A.1.1 COUNTSAT Problem

COUNTSAT [77] is an instance of the MAXSAT problem. In COUNTSAT, the value of a given solution is the number of satisfied clauses (among all the possible Horn clauses of three variables) by an input composed by n boolean variables. It is easy to check that the optimum is obtained when the value of all the variables is 1. In this study we consider an instance of $n = 20$ variables, and the value of the optimal solution to the problem is 6860 (Eq. A.2).

$$f_{\text{COUNTSAT}}(s) = s + n(n-1)(n-2) - 2(n-2)\binom{s}{2} + 6\binom{s}{3} \ ,$$
$$= s + 6840 - 18s(s-1) + s(s-1)(s-2) \ . \quad \text{(A.2)}$$

COUNTSAT is obtained from MAXSAT with the intention of being very difficult to solve for GAs [77]. The values assigned to the variables randomly following a uniform distribution should be composed by approximately $n/2$ ones in average. Thus, local changes decreasing the number of ones will lead us to better results, while incrementing the number of ones will decrement the fitness value (see Fig. A.1). Consequently, it is expected that the GA rapidly finds the solution composed by all the genes set to 0 and has difficulties for finding the optimal solution, composed by ones in all its positions.

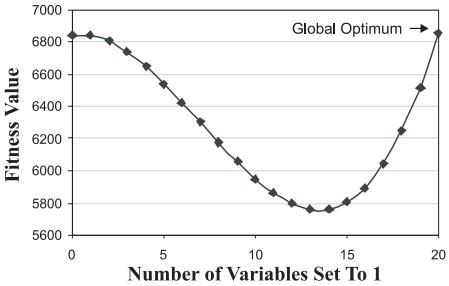


Fig. A.1. COUNTSAT function with 20 variables

A.1.2 Error Correcting Codes Design Problem – ECC

The ECC problem was presented in [173]. We 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. Our objective will be to find a code which has a value for d as large as possible (reflecting greater tolerance to noise and errors), given previously fixed values for n and M. The fitness function to be maximized is:

$$f_{\text{ECC}}(C) = \frac{1}{\sum_{i=1}^{M} \sum_{\substack{j=1 \\ i \neq j}}^{M} \frac{1}{d_{ij}^2}} \ , \quad \text{(A.3)}$$

where d_{ij} represents the Hamming distance between codewords i and j in the code (made up of M codewords, each of length n). We consider in the present book an instance where $M = 24$ and $n = 12$. The size of the search space is approximately 10^{87}. The optimal solution for this problem has a fitness value of 0.0674 [45]. In some of our studies, we simplified the search space of the problem reducing to the half $(M/2)$ the number of words composing the code, and the other half is composed by the complement of the words found by the algorithm.

A.1.3 Frequency Modulation Sounds – FMS

The FMS problem [251] is defined as determining the 6 real parameters $\mathbf{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ of the frequency modulated sound model given in Eq. A.4 for approximating it to the sound wave given in Eq. A.5 (where $\theta = 2 \cdot \pi/100$). The problem can be defined as a discrete or continuous optimization problem. In the discrete case, the parameters are defined in the range $[-6.4, +6.35]$, and we encode each parameter into a 32 bit substring in the individual. The continuous case is described in Sect. A.2.

$$y(t) = a_1 \sin\Big(w_1 t\theta + a_2 \sin\big(w_2 t\theta + a_3 \sin(w_3 t\theta)\big)\Big) \ , \tag{A.4}$$

$$y_0(t) = 1.0 \sin\Big(5.0 t\theta - 1.5 \sin\big(4.8 t\theta + 2.0 \sin(4.9 t\theta)\big)\Big) \ . \tag{A.5}$$

The goal is to minimize the sum of square errors between the sample data (Eq. A.4) and the real (Eq. A.5), as it is given in detailed in Eq. A.6. This problem is a highly complex multimodal function having strong epistasis. Due to the high difficulty of solving this problem with high accuracy without applying local search or specific operators for continuous optimization (like gradual GAs [127]), we stop the algorithm when the error falls below 10^{-2}. The fitness function to maximize corresponds with the inverse of the function shown in Eq. A.6, and we obtain the maximum value when $E_{FMS} = 0.0$.

$$E_{\text{FMS}}(\mathbf{x}) = \sum_{t=0}^{100} \Big(y(t) - y_0(t)\Big)^2 \ . \tag{A.6}$$

A.1.4 IsoPeak Problem

IsoPeak is a non separable problem investigated in [175]. The solutions for this function are composed by an n-dimensional vector, where $n = 2 \times m$ (the genes are divided in groups of two). We first define two auxiliar functions $Iso1$ and $Iso2$ as:

$\overrightarrow{x}$	**00**	**01**	**10**	**11**
$Iso1$	m	0	0	$m-1$
$Iso2$	0	0	0	m

Now, we can define the IsoPeak problem as:

$$f_{\text{IsoPeak}(\overrightarrow{x})} = Iso2\,(x_1, x_2) + \sum_{i=2}^{m} Iso1\,(x_{2i-1}, x_{2i}) \ . \tag{A.7}$$

The aim of the problem is to maximize the function f_{IsoPeak} and the global optimum is reached when the values of the variables of the n-dimensional vector are $(1, 1, 0, 0, \ldots, 0, 0)$.

A.1.5 Maximum Cut of a Graph – MAXCUT

The MAXCUT problem lies in dividing a weighted graph $G = (V, E)$ into two disjoint subsets $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ so that the sum of the weights of the edges with one endpoint in V_0 and the other one in V_1 is maximized. For encoding the problem we use a binary string $(x_1, x_2, \ldots, x_n)$ where each digit corresponds to a vertex. If a digit is 1 then the corresponding vertex is in set V_1; if it is 0 then the corresponding vertex is in set V_0. The function to be maximized [144] is:

$$f_{\text{MAXCUT}}(\mathbf{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \cdot \left[x_i(1 - x_j) + x_j(1 - x_i) \right] \ . \tag{A.8}$$

Note that w_{ij} contributes to the sum only if nodes i and j are in different partitions. Although it is possible to generate different instances of graph randomly, we have used three different instances of the problem taken from [144]. Two of them are randomly generated graphs of moderate sizes: a sparse one MAXCUT20.01, and a dense one MAXCUT20.09; both of them are made up of 20 vertices. The other instance is a scalable weighted graph of 100 vertices. The globally optimal solutions for these instances are 10.119812 for MAXCUT20.01, 56.740064 in the case of MAXCUT20.09, and 1077 for MAXCUT100.

A.1.6 Massively Multimodal Deceptive Problem – MMDP

The MMDP is a problem that has been specifically designed to be difficult for an EA [107]. It is made up of k deceptive subproblems (s_i) of 6 bits each one. The value of each of these sub-problems (fitness_{s_i}) depends on the number of ones a binary string has (see Fig. A.2). It is easy to understand why these functions are considered deceptive, as they have two global maxima and a deceptive attractor in the middle point (see graphic of Fig. A.2).

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

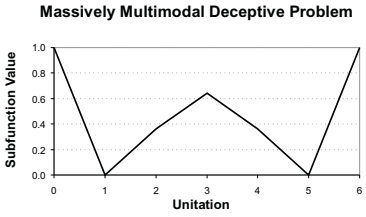


Fig. A.2. Basic deceptive bipolar function (s_i) for MMDP

In MMDP, each subproblem s_i contributes to the total fitness value according to the number of ones it has (Fig. A.2). The global optimum has a value of k and it is attained when every subproblem is composed of zero 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 k parameter. We use here a considerably large instance (if the contrary is not specified) of $k = 40$ subproblems. The instance we try to maximize for solving the problem is shown in Eq. A.9, and its maximum value is 40.

$$f_{\text{MMDP}}(\mathbf{s}) = \sum_{i=1}^{k} \text{fitness}_{s_i} \ . \tag{A.9}$$

A.1.7 Minimum Tardy Task Problem – MTTP

MTTP [236] is a task-scheduling problem wherein each task i from the set of tasks $T = \{1, 2, \ldots, n\}$ has a length l_i (the time it takes for its execution), a deadline d_i (before which a task must be scheduled), and a weight w_i. The weight is a penalty that has to be added to the objective function in the event that the task remains unscheduled. The lengths, weights and deadlines of tasks are all positive integers. Scheduling the tasks of a subset S of T is to find the starting time of each task in S, such as at most one task at time is performed and such that each task finishes before its deadline.

We characterize a scheduling function g defined on a subset of tasks $S \subseteq T : S \mapsto \mathbb{Z}^+ \bigcup \{0\} \mid \forall\ i,\ j \in S$ with the following two properties:

1. A task can not be scheduled before any previous one has finished: $g(i) < g(j) \Rightarrow g(i) + \ l_i \leq g(j)$.
2. Every task finishes before its deadline: $g(i) + \ l_i \leq d_i$.

The objective function for this problem is to minimize the sum of the weights of the unscheduled tasks as shown in Eq. A.10. Therefore, the optimum scheduling minimizes the function:

$$W(\mathbf{x}) = \sum_{i \in T - S} w_i \ . \tag{A.10}$$

The schedule of tasks S can be represented by a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ containing all the tasks ordered by its deadline. Each $x_i \in \{0, 1\}$, where if $x_i = 1$ then task i is scheduled in S, while if $x_i = 0$ means that task i is not included in S. The fitness function to optimize, described in [144], is the inverse of Eq. A.10: $f_{\text{MTTP}}(\mathbf{x}) = 1/W(\mathbf{x})$. We have used in this study three different instances [144]: MTTP20, MTTP100, and MTTP200, with sizes 20, 100 and 200, and known maximum fitness values of 0.02439, 0.005 and 0.0025, respectively.

A.1.8 OneMax Problem

The OneMax problem [222] is a very simple problem consisting in maximizing the number of ones contained in a bit string. Formally, this problem can be described as finding a string $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \{0, 1\}$, which maximizes the following equation:

$$f_{\text{OneMax}}(\mathbf{x}) = \sum_{i=1}^{n} x_i \ . \tag{A.11}$$

The function f_{OneMax} has $(n + 1)$ possible different values. For the *additionally decomposable functions* (ADFs) the multimodal distribution occurs quite frequently [108]. For defining an instance of this problem, we only need to define the length of the bit string (n). For a given n, the optimal solution to the problem is a string of n ones, that is, the value of all the bits of the string is one.

A.1.9 Plateau Problem

This problem was studied in [188], and it is also known as the real path of three bits problem. The solutions for this function consist of a n-dimensional vector, where $n = 3 \times m$ (the genes are divided in groups of three). We first define an auxiliar function g as:

$$g(x_1, x_2, x_3) = \begin{cases} 1, & \text{if} \quad x_1 = 1 \quad \text{and} \quad x_2 = 1 \quad \text{and} \quad x_3 = 1 \\ 0, & \text{otherwise} \ . \end{cases} \tag{A.12}$$

Now, we can define the Plateau problem as:

$$f_{\text{Plateau}}(\mathbf{x}) = \sum_{i=1}^{m} g(\mathbf{s_i}) \ . \tag{A.13}$$

Where $\mathbf{s_i} = (x_{3i-2}, x_{3i-1}, x_{3i})$. The aim of the problem is to maximize the function f_{Plateau}, and the global optimum is obtained when all the bits of the string are set to one.

A.1.10 P-PEAKS Problem

The P-PEAKS problem [140] is a multimodal problem generator. A problem generator is an easily parameterizable task which has a tunable degree of difficulty, so that we can generate instances as complex as we want. Also, using a problem generator removes the opportunity to hand-tune algorithms to a particular problem, therefore allowing a larger fairness when comparing algorithms. With a problem generator we evaluate our algorithms on a high number of random problem instances, since a different instance is solved each time the algorithm runs, then the predictive power of the results for the problem class as a whole is increased, not for particular instances.

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 hamming distance between this string and the closest peak, divided by N (as shown in Eq. A.14). Using a higher (or lower) number of peaks we obtain more (or less) epistatic problems. In this book we have used an instance of $P = 100$ peaks of length $N = 100$ bits each, which represents a medium/high difficulty level [26]. The maximum fitness value for this problem is 1.0.

$$f_{\text{P-PEAKS}}(\mathbf{x}) = \frac{1}{N} \max_{i=1}^{P} \{N - \text{Hamming}(\mathbf{x}, \text{Peak}_i)\} \ . \tag{A.14}$$

A.1.11 Satisfiability Problem – SAT

The satisfiability problem (SAT) has received much attention by the scientific community since it plays a main role in the NP-completeness of the problems [96]. This is due to it was demonstrated that any NP problem can be translated into an equivalent SAT problem in polynomial time (Cook theorem) [51], while the inverse transformation may not always exist in polynomial time. The SAT problem was the first which was demonstrated to belong to the NP class of problems.

The SAT problem consists in assigning values to a set of n boolean variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ such that they satisfy a given set of clauses $c_1(\mathbf{x}), \ldots, c_m(\mathbf{x})$, where $c_i(\mathbf{x})$ is a disjunction of literals, and a literal is a variable or its negation. Hence, we can define SAT as a function $f : \mathbb{B}^n \rightarrow \mathbb{B}$, $\mathbb{B} = \{0, 1\}$ like:

$$f_{\text{SAT}}(\mathbf{x}) = c_1(\mathbf{x}) \wedge c_2(\mathbf{x}) \wedge \ldots \wedge c_m(\mathbf{x}) \ . \tag{A.15}$$

An instance of SAT, $\mathbf{x}$, is called satisfiable if $f_{SAT}(\mathbf{x}) = 1$, and unsatisfiable otherwise. A k-SAT instance is composed of clauses with length k, and when $k \geq 3$ the problem is NP-complete [96]. In this book we only consider 3-SAT instances belonging to the difficulty transition step of the SAT problem, where the hardest instances belong to, and verifying that $m = 4.3 * n$ [182] (being m the number of clauses and n the number of variables). Particulary, in this book we used sets of 12 instances (with $n = 30$ to 100 variables) proposed in [32], belonging to the difficulty transition step of the SAT problem.

According to Eq. A.15, a simple fitness function for SAT consists of counting the number of clauses that satisfies the evaluating solution. The problem is that this function assigns the same fitness value to multiple different solutions. An alternative lies in considering the fitness function as a lineal function of the number of satisfied clauses where the *stepwise adaptation of weights* (SAW) [83] is used:

$$f_{\text{SAW}}(\mathbf{x}) = w_1 \cdot c_1(\mathbf{x}) + \ldots + w_m \cdot c_m(\mathbf{x}) \ . \tag{A.16}$$

This function weights the values of the clauses with $w_i \in \mathbb{N}$ in order to give more importance to those clauses which are not satisfied yet by the current best solution. These weights are adjusted dynamically according to $w_i = w_i + 1 - c_i(\mathbf{z})$, being $\mathbf{z}$ the current fittest individual.

A.2 Continuous Optimization Problems

In this section we present the problems belonging to the continuous optimization field studied in this book. Some of these problems are well known classical academic functions (Sect. A.2.1), whereas others are taken from the real world (Sect. A.2.2).

A.2.1 Academic Problems

As stated before, eight academic problems for continuous optimization are presented in this section. *Griewangk* (f_{Gri}) [124], *Rastrigin generalized* (f_{Ras}) [30, 247], *Rosenbrock generalized* (f_{Ros}) [63], *Schwefel 1.2* (f_{Sch}) [226], the *Sphere* model (f_{Sph}) [63, 226], the *expansion of* f_{10} (ef_{10}) [260], the *Fractal* (f_{Frac}) [31], and the *Ackley* functions (f_{Ack}), originally proposed by Ackley [3] as a bidimensional function, and later generalized by Bäck et al. [34]. This set of problems incorporates very diverse characteristics, since it is composed by lineal and non lineal, unimodal and multimodal, scalable and non scalable, convex, etc. problems. All the details on these problems are shown in Table A.1, where n is the size of the problem we use in this work. The value of the optimal solution for all the problems of Table A.1 is 0.0.

Table A.1. Academic benchmark of continuous optimizaton

Problem	Fitness Function	Variable Values	n
Griewangk	$f_{\text{Gri}}(\mathbf{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ $d = 4000$	$-600.0 \leq x_i \leq 600.0$	25
Rastrigin	$f_{\text{Ras}}(\mathbf{x}) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i)$ $a = 10,\ \omega = 2\pi$	$-5.12 \leq x_i \leq 5.12$	25
Rosenbrock	$f_{\text{Ros}}(\mathbf{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$-5.12 \leq x_i \leq 5.12$	25
Schwefel	$f_{\text{Sch}}(\mathbf{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	$-65.536 \leq x_i \leq 65.536$	25
Sphere	$f_{\text{Sph}}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$-5.12 \leq x_i \leq 5.12$	25
ef_{10}	$f_{\text{ef}_{10}}(\mathbf{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{i-1}, x_i) + \ldots + f_{10}(x_n, x_1)$ $f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot [\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1]$	$-100.0 < x_i \leq 100.0$	10
Fractal	$f_{\text{Frac}}(\mathbf{x}) = \sum_{i=1}^{n}(C'(x_i) + x_i^2 - 1)$ $C'(z) = \begin{cases} \frac{C(z)}{C(1)\lvert z\rvert^{2-D}} & \text{si } z \neq 0 \\ 1 & \text{si } z = 0 \end{cases}$ $C(z) = \sum_{j=-\infty}^{\infty} \frac{1-\cos(b^j z)}{b^{(2-D)j}}$ $D = 1.85,\ b = 1.5$	$-5.00 \leq x_i \leq 5.00$	20
Ackley	$f_{\text{Ack}}(\mathbf{x}) = -a\exp\left[-b\left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2}\right] - -exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + e$ $a = 20,\ b = 0.2,\ c = 2\pi$	$-32.768 \leq x_i \leq 32.768$	25

During our tests, we have noticed that there exists an important loose of accuracy when computing the Griewangk's and Rastrigin's functions when the solution is near the optimal value. The reason is that when computing the value of the functions for a solution close to the optimal one (allele values very close to 0.0), there are some addends in the functions ($\frac{1}{d}\sum_{i=1}^{n} x_i^2$ in f_{Gri} and $\sum_{i=1}^{n} x_i^2$ in f_{Ras}) that are rounded to 0.0 when computing the functions because they are more than 100 orders of magnitude smaller than the other addends. In order to avoid that undesired rounding we propose in this paper two new definitions for those functions which allow us to obtain higher levels of accuracy, and they simply consist in altering the order of the addends in the function —see Eqs. A.17 and A.18. However, we study in this paper the same (less accurate) versions of the functions existing in the literature in order to being fair in our comparisons.

$$f_{\text{Gri\_Acc}}(\mathbf{x}) = \left(1 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + \frac{1}{d}\sum_{i=1}^{n} x_i^2 \ ; \tag{A.17}$$

$$f_{\text{Ras\_Acc}}(\mathbf{x}) = \left(a \cdot n - \sum_{i=1}^{n} a \cdot \cos(\omega \cdot x_i)\right) + \sum_{i=1}^{n} x_i^2 \ . \tag{A.18}$$

Now, we present the differences on the results when using these two fitness functions for each problem (the accurate or the standard one) with JCell (the parametrization used is shown in Chap. 12). The comparison of the results of executing JCell on these functions is shown in Table A.2. As it can be seen, when we use the proposed accurate functions, the algorithm does not find the optimum in any of the runs, while in the case of standard functions (using the same parametrization) it is found in more than 80% of the runs for both problems. Additionally, after computing the p-values on the results, we obtained statistic relevance in all the differences, although the average values showed are similar in the two versions of the problems.

Table A.2. Comparison between the results obtained with the Griewangk and Rastrigin standard functions and their new more accurate definitions

JCell	Best	Average	Test
f_{Gri}	94%	2.381e-3 $\pm 4.750e{-3}$	+
$f_{\text{Gri\_Acc}}$	4.13e-156	2.300e-3 $\pm 4.750e{-3}$	
f_{Ras}	82%	2.734e-5 $\pm 9.299e{-6}$	+
$f_{\text{Ras\_Acc}}$	2.734e-138	2.113e-6 $\pm 9.299e{-6}$	

A.2.2 Real World Problems

We include in our studies three real world problems for better assessing our conclusions. These problems are the *frequency modulation sound parameter identification problem* (f_{fms}) [250], *systems of linear equations* (f_{Sle}) [85] and a *polynomial fitting problem* (f_{Cheb}) [238].

Frequency modulation sound parameter identification problem

This problem is defined as determining the 6 real parameters $\mathbf{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ of the frequency modulated sound model given in Eq. A.19 for approximating it to the sound wave given in Eq. A.20 (where $\theta = 2 \cdot \pi/100$). The parameters are defined in the range $[-6.4, +6.35]$.

$$y(t) = a_1 \cdot \sin(\omega_1 \cdot t \cdot \theta + a_2 \cdot \sin(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin(\omega_3 \cdot t \cdot \theta))) \ , \tag{A.19}$$

$$y_0(t) = 1.0 \cdot \sin(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin(4.9 \cdot t \cdot \theta))) \ . \tag{A.20}$$

The goal is to minimize the sum of square errors given by Eq. A.21. This problem is a highly complex multimodal function having strong epistasis, with optimum value 0.0.

$$f_{\text{fms}}(\mathbf{x}) = \sum_{t=0}^{100} \Big(y(t) - y_0(t) \Big)^2 \ . \tag{A.21}$$

Systems of linear equations

This problem lies in finding the values for the vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$, with:

$$A = \begin{pmatrix} 5,4,5,2,9,5,4,2,3,1 \\ 9,7,1,1,7,2,2,6,6,9 \\ 3,1,8,6,9,7,4,2,1,6 \\ 8,3,7,3,7,5,3,9,9,5 \\ 9,5,1,6,3,4,2,3,3,9 \\ 1,2,3,1,7,6,6,3,3,3 \\ 1,5,7,8,1,4,7,8,4,8 \\ 9,3,8,6,3,4,7,1,8,1 \\ 8,2,8,5,3,8,7,2,7,5 \\ 2,1,2,2,9,8,7,4,4,1 \end{pmatrix} ; \ \mathbf{b} = \begin{pmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{pmatrix} \ . \tag{A.22}$$

The evaluation function we minimize in our experiments is shown in Eq. A.23. This function has the optimal solution $f_{\text{sle}}(\mathbf{x}^*) = 0.0$, and the values of the ten variables of the problem range into the interval $[-9.0, 11.0]$.

$$f_{\text{Sle}}(\mathbf{x}) = \left| \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} \cdot x_j) - b_i \right| \ . \tag{A.23}$$

Polynomial fitting problem

For this problem, the objective is to find the coefficients of the following polynomial in z:

$$P(z) = \sum_{j=0}^{2k} c_j z^j, \quad k \in \mathbb{Z}^+ \ , \tag{A.24}$$

so that $\forall z^j \in [-1, +1]$, $P(z) \in [-1, +1]$, $P(+1.2) \geq T_{2k}(+1.2)$, and $P(-1.2) \geq T_{2k}(-1.2)$, where $T_{2k}(z)$ is a f_{Cheb} polynomial of degree $2k$.

The solution to the polynomial fitting problem consists of the coefficients of $T_{2k}(z)$. This polynomial oscillates between -1 and +1. Outside this region, the polynomial rises steeply in the direction of high positive ordinate values. This problem has its roots in electronic filter design, and it challenges an optimization procedure by forcing it to find parameter values with grossly different magnitudes, something that is very common in industrial systems. The f_{Cheb} polynomial employed here is:

$$T_8(z) = 1 - 32z^2 + 160z^4 - 256z^6 + 128z^8 \ . \tag{A.25}$$

It is a nine-parameter problem ($\mathbf{x} = [x_1, \ldots, x_9]$). A small correction is needed in order to transform the constraints of this problem into an objective function to be minimized, called f_{Cheb} (see [127] for all the details). Each parameter (coefficient) is in the range [-5.12,+5.12]. The objective function value of the optimum is $f_{\text{Cheb}}(\mathbf{x}^*) = 0.0$.

A.3 Multi-objective Optimization Problems

In this section we present the theoretical multi-objective benchmark studied. We can divide it into constrained and non constrained problems. The non constrained problems chosen include the Schaffer, Fonseca, and Kursawe, and also the ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 [270] problems. The formulation is shown in Table A.3. The constrained problems are Osyczka2, Tanaka, Srinivas, and ConstrEx, and they are described in Table A.4. All these problems are well known in the field of the multi-objective optimization, and can be found in books as [49, 65].

Another benchmark used in this study is the one obtained using WFG, a multi-objective problem generator tool recently proposed in [136]. This tool allows the user to define benchmarks with different characteristics. In this study we use a benchmark of 9 bi-objective problems, from WFG1 to WFG9. The properties of these problems are detailed in Table A.5.

Table A.3. Non constrained test functions

Problem	Objective Functions	Variable Values	n
Schaffer	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10^5 \leq x \leq 10^5$	1
Fonseca	$f_1(\mathbf{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2}$ $f_2(\mathbf{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2}$	$-4 \leq x_i \leq 4$	3
Kursawe	$f_1(\mathbf{x}) = \sum_{i=1}^{n-1}\left(-10e^{\left(-0.2*\sqrt{x_i^2+x_{i+1}^2}\right)}\right)$ $f_2(\mathbf{x}) = \sum_{i=1}^{n}(\lvert x_i\rvert^a + 5\sin x_i^b); a = 0.8; b = 3$	$-5 \leq x_i \leq 5$	3
ZDT1	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = g(\mathbf{x})[1 - \sqrt{x_1/g(\mathbf{x})}]$ $g(\mathbf{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT2	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = g(\mathbf{x})[1 - (x_1/g(\mathbf{x}))^2]$ $g(\mathbf{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT3	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = g(\mathbf{x})\left[1 - \sqrt{\frac{x_1}{g(\mathbf{x})}} - \frac{x_1}{g(\mathbf{x})}\sin(10\pi x_1)\right]$ $g(\mathbf{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT4	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = g(\mathbf{x})[1 - (x_1/g(\mathbf{x}))^2]$ $g(\mathbf{x}) = 1 + 10(n-1) + \sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	$0 \leq x_1 \leq 1$ $-5 \leq x_i \leq 5$ $i = 2, ..., n$	10
ZDT6	$f_1(\mathbf{x}) = 1 - e^{-4x_1}\sin^6(6\pi x_1)$ $f_2(\mathbf{x}) = g(\mathbf{x})[1 - (f_1(\mathbf{x})/g(\mathbf{x}))^2]$ $g(\mathbf{x}) = 1 + 9[(\sum_{i=2}^{n} x_i)/(n-1)]^{0.25}$	$0 \leq x_i \leq 1$	10

Table A.4. Constrained test functions

Problem	Objective functions	Constraints	Variable Values	n
Osyczka2	$f_1(\mathbf{x}) = -(25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2(x_4-4)^2 + (x_5-1)^2)$ $f_2(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$g_1(\mathbf{x}) = 0 \leq x_1 + x_2 - 2$ $g_2(\mathbf{x}) = 0 \leq 6 - x_1 - x_2$ $g_3(\mathbf{x}) = 0 \leq 2 - x_2 + x_1$ $g_4(\mathbf{x}) = 0 \leq 2 - x_1 + 3x_2$ $g_5(\mathbf{x}) = 0 \leq 4 - (x_3-3)^2 - x_4$ $g_6(\mathbf{x}) = 0 \leq (x_5-3)^3 + x_6 - 4$	$0 \leq x_1, x_2 \leq 10$ $1 \leq x_3, x_5 \leq 5$ $0 \leq x_4 \leq 6$ $0 \leq x_6 \leq 10$	6
Tanaka	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = x_2$	$g_1(\mathbf{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(x_1/x_2)) \leq 0$ $g_2(\mathbf{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$	$-\pi \leq x_i \leq \pi$	2
ConstrEx	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = (1 + x_2)/x_1$	$g_1(\mathbf{x}) = x_2 + 9x_1 \geq 6$ $g_2(\mathbf{x}) = -x_2 + 9x_1 \geq 1$	$0.1 \leq x_1 \leq 1.0$ $0 \leq x_2 \leq 5$	2
Srinivas	$f_1(\mathbf{x}) = (x_1-2)^2 + (x_2-1)^2 + 2$ $f_2(\mathbf{x}) = 9x_1 - (x_2-1)^2$	$g_1(\mathbf{x}) = x_1^2 + x_2^2 \leq 225$ $g_2(\mathbf{x}) = x_1 - 3x_2 \leq -10$	$-20 \leq x_i \leq 20$	2

Table A.5. Properties of MOPs created with WFG

Problem	Separability	Modality	bias	Geometry
WFG1	separable	uni	polynomial, plain	convex, mixed
WFG2	non separable	f_1 uni, f_2 multi	do not have	convex, desconected
WFG3	non separable	uni	do not have	linear, degenerated
WFG4	non separable	multi	do not have	concave
WFG5	separable	deceptive	do not have	concave
WFG6	non separable	uni	do not have	concave
WFG7	separable	uni	dependent on the parameter	concave
WFG8	non separable	uni	dependent on the parameter	concave
WFG9	non separable	multi, deceptive	dependent on the parameter	concave

References

1. Java language specification. http://java.sun.com.
2. ProActive official web site. http://www-sop.inria.fr/oasis/proactive/.
3. D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston, MA, 1987.
4. P. Adamidis and V. Petridis. Co-operating populations with different evolution behaviours. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 188–191. IEEE Press, 1996.
5. E. Alba. *Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos*. PhD thesis, Universidad de Málaga, Málaga, Febrero 1999.
6. E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, October 2005.
7. E. Alba, H. Alfonso, and B. Dorronsoro. Advanced models of cellular genetic algorithms evaluated on SAT. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 1123–1130, Washington D.C., USA, June 25–29 2005. ACM Press.
8. E. Alba, P. Bouvry, B. Dorronsoro, F. Luna, and A.J. Nebro. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. In *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) Workshop*, page 192b, Denver, Colorado, USA, 2005.
9. E. Alba, J.F. Chicano, B. Dorronsoro, and G. Luque. Diseño de códigos correctores de errores con algoritmos genéticos. In *Actas del Tercer Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 51–58, Córdoba, Spain, 2004.
10. E. Alba and B. Dorronsoro. Auto-adaptación en algoritmos evolutivos celulares. Un nuevo enfoque algorítimico. In *Actas del Segundo Congreso Español sobre Metahurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 176–185, Gijón, Spain, 2003.
11. E. Alba and B. Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 3004 of *Lecture Notes in Computer Science (LNCS)*, pages 11–20, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag, Heidelberg.

12. E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.
13. E. Alba and B. Dorronsoro. Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, 98(6):225–230, June 2006.
14. E. Alba and B. Dorronsoro. *Engineering Evolutionary Intelligent Systems*, chapter 13, A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem. Studies in Computational Intelligence. Springer-Verlag, Heidelberg, 2007. To appear.
15. E. Alba, B. Dorronsoro, and H. Alfonso. Cellular memetic algorithms. *Journal of Computer Science and Technology*, 5(4):257–263, December 2005.
16. E. Alba, B. Dorronsoro, and H. Alfonso. Cellular memetic algorithms evaluated on SAT. In *XI Congreso Argentino de Ciencias de la Computación (CACIC)*, 2005. CD Edition.
17. E. Alba, B. Dorronsoro, M. Giacobini, and M. Tomassini. *Handbook of Bioinspired Algorithms and Applications*, chapter 7, Decentralized Cellular Evolutionary Algorithms, pages 103–120. CRC Press, 2006.
18. E. Alba, B. Dorronsoro, F. Luna, A.J. Nebro, P. Bouvry, and L. Hogie. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. *Computer Communications*, 30(4):685–697, 2007.
19. E. Alba, M. Giacobini, M. Tomassini, and S. Romero. Comparing synchronous and asynchronous cellular genetic algorithms. In J.J. Merelo et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature VII (PPSN-VII)*, volume 2439 of *Lecture Notes in Computer Science (LNCS)*, pages 601–610, Granada, Spain, 2002. Springer-Verlag, Heidelberg.
20. E. Alba, F. Luna, and A.J. Nebro. Advances in parallel heterogeneous genetic algorithms for continuous optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3):101–117, 2004.
21. E. Alba and G. Luque. Growth curves and takeover time in distributed evolutionary algorithms. In K. Deb et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 3102 of *Lecture Notes in Computer Science (LNCS)*, pages 864–876. Springer-Verlag, Heidelberg, 2004.
22. E. Alba and G. Luque. A new local search algorithm for the DNA fragment assembly problem. In C. Cotta and J. van Hemert, editors, *Evolutionary Computation in Combinatorial Optimization (Evo*)*, volume 4446 of *Lecture Notes in Computer Science (LNCS)*, pages 1–12, Valencia, Spain, April 2007. Springer-Verlag, Heidelberg.
23. E. Alba, J. Madera, B. Dorronsoro, A. Ochoa, and M. Soto. Theory and practice of cellular UMDA for discrete optimization. In T.P. Runarsson et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature IX (PPSN-IX)*, volume 4193 of *Lecture Notes in Computer Science (LNCS)*, pages 242–251, Reykjavik, Iceland, September 2006. Springer-Verlag, Heidelberg.
24. E. Alba and J.F. Saucedo. Panmictic versus decentralized genetic algorithms for non-stationary problems. In *Sixth Metaheuristics International Conference (MIC)*, pages 7–12, Austria, 2005.
25. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

26. E. Alba and J.M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer, editor, *Proc. of the International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, volume 1917 of *Lecture Notes in Computer Science (LNCS)*, pages 29–38. Springer-Verlag, Heidelberg, 2000.
27. E. Alba and J.M. Troya. Improving flexibility and efficiency by adding parallelism to genetic algorithms. *Statistics and Computing*, 12(2):91–114, 2002.
28. J. Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS - a genetic algorithm with varying population size. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 1, pages 73–78, 1994.
29. P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Research Report 949-M, Universite Joseph Fourier, Grenoble, France, 1995.
30. T. Bäck. Self-adaptation in genetic algorithms. In F.J. Varela and P. Bourgine, editors, *Proc. of the 1st European Conference on Artificial Life*, pages 263–271, Cambridge, MA, 1992. The MIT Press.
31. T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, 1996.
32. T. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *International Conference on Evolutionary Programming VII*, volume 1477 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Heidelberg, 1998.
33. T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.
34. T. Bäck, G. Rudolf, and H.-P. Schwefel. Evolutionary programming and evolution strategies: similarities and differences. In D.B. Fogel and W. Atmar, editors, *Proc. of the Second Conference on Evolutionary Programming*, pages 11–22, La Jolla, California, 1993. Evolutionary Programming Society.
35. S. Baluja. Structure and performance of fine-grain parallelism in genetic search. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 155–162. Morgan Kaufmann, 1993.
36. W. Banzhaf. The "molecular" traveling salesman. *Biological Cybernetics*, 64:7–14, 1990.
37. J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In E. Cantú-Paz, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 2723 of *Lecture Notes in Computer Science (LNCS)*, pages 646–656, Illinois, Chicago, USA, 2003. Springer-Verlag, Heidelberg.
38. A. Bethke. Comparison of genetic algorithms and gradient based optimizers on parallel processors: Efficiency of use of precessing capacity. Technical Report 197, Ann Arbor, University of Michigan, 1976.
39. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
40. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, 1999.
41. H.J. Bremermann. *Self-Organizing Systems*, chapter Optimization Trough Evolution and Resombination, pages 93–106. Spartan Books, Washington DC, 1962.

42. S. Cahon, N. Melab, and E-G. Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004.
43. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*, volume 1 of *Book Series on Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2nd edition, 2000.
44. 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.
45. 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.
46. N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *Operations Research Quartely*, 20:309–318, 1969.
47. N. Christofides, A. Mingozzi, and P. Toth. *Combinatorial Optimization*, chapter The Vehicle Routing Problem, pages 315–338. John Wiley & Sons, 1979.
48. C.A. Coello and G. Toscano. Multiobjective optimization using a micro-genetic algorithm. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 274–282, 2001.
49. C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.
50. R.J. Collins and D.R. Jefferson. Selection in massively parallel genetic algorithms. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 249–256, San Diego, CA, USA, 1991. Morgan Kaufmann.
51. S.A. Cook. The complexity of theorem-proving procedures. *Proc. of the Third Anual ACM Symp. on the Theory of Computing*, pages 151–158, 1971.
52. J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. *Logistics Systems: Design and Optimization*, chapter 9. New Heuristics for the Vehicle Routing Problem, pages 279–298. Springer-Verlag, Heidelberg, 2004.
53. C. Cotta, E. Alba, and J.M. Troya. Un estudio de la robustez de los algoritmos genéticos paralelos. *Revista Iberoamericana de Inteligencia Artificial*, 98(5):6–13, 1998.
54. N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proc. of the First International Conference on Genetic Algorithms and their Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, July 24–26 1985.
55. G.A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.
56. G.B. Dantzing and R.H. Ramster. The truck dispatching problem. *Management Science*, 6:80–91, 1959.
57. C. Darwin. *On the Origin of Species by Means of Natural Selection*. John Murray, Londres, 1859.
58. Y. Davidor. A naturally occurring niche and species phenomenon: The model and first results. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 257–263, San Diego, CA, July 1991. Morgan Kaufmann.

59. Y. Davidor, T. Yamada, and R. Nakano. The ECOlogical framework II: Improving GA performance at virtually zero cost. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 171–176. Morgan Kaufmann, 1993.
60. L. Davis. Adapting operator probabilities in genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 61–69. Morgan Kaufmann, 1989.
61. L. Davis. *Handbook of genetic algorithms*, volume 1991. Van Nostrand Reinhold, New York.
62. K. De Jong and J. Sarma. On decentralizing selection algorithms. In L. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms (ICGA)*, pages 17–23, San Francisco, CA, 1995. Morgan Kaufmann.
63. K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975. Ann Arbor.
64. K.A. De Jong and W.M. Spears. Using genetic algorithm to solve NP-complete problems. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 124–132. Morgan Kaufmann, 1989.
65. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.
66. K. Deb and R.B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.
67. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
68. G. Dick. A comparison of localised and global niching methods. In *Annual Colloquium of the Spatial Information Research Centre (SIRC)*, pages 91–101, Dunedin, New Zealand, Novenber 2005.
69. G. Dick and P. Whigham. The behaviour of genetic drift in a spatially-structured evolutionary algorithm. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 2, pages 1855–1860. IEEE Press, 2005.
70. M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, 2004.
71. B. Dorronsoro. The VRP web, http://neo.lcc.uma.es/radi-aeb/WebVRP.
72. B. Dorronsoro. *Diseño e Implementación de Algoritmos Genéticos Celulares para Problemas Complejos*. PhD thesis, University of Málaga, February 2007.
73. B. Dorronsoro and E. Alba. A simple cellular genetic algorithm for continuous optimization. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 2838–2844, Vancouver, Canada, July 2006. IEEE Press.
74. B. Dorronsoro, E. Alba, M. Giacobini, and M. Tomassini. The influence of grid shape and asynchronicity on cellular evolutionary algorithms. In Y. Shi, editor, *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 2152–2158, Portland, Oregon, June 20–23 2004. IEEE Press.
75. B. Dorronsoro, D. Arias, F. Luna, A.J. Nebro, and E. Alba. A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. In Waleed W. Smari, editor, *High Performance Computing & Simulation Conference (HPCS)*, pages 759–765, 2007.
76. B. Dorronsoro, A.J. Nebro, D. Arias, and E. Alba. Un algoritmo genético híbrido paralelo para instancias complejas del problema VRP. In *Actas*

del Quinto Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pages 135–141, Puerto de la Cruz, Tenerife, Spain, 2007.
77. S. Droste, T. Jansen, and I. Wegener. A natural and simple function which is hard for all evolutionary algorithms. In *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, pages 2704–2709, Nagoya, Japan, 2000.
78. T. Duncan. Experiments in the use of neighbourhood search techniques for vehicle routing. Technical Report AIAI-TR-176, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, 1995.
79. J.J. Durillo. jMetal framework, http://neo.lcc.uma.es/software/metal/.
80. J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: A java framework for developing multiobjective optimization metaheuristics. Technical Report ITI-2006-10, Dpto. de Lenguajes y CC.CC., Universidad de Málaga, 2006.
81. E. Alba and the MALLBA Group. MALLBA: A library of skeletons for combinatorial optimization. In R.F.B. Monien, editor, *Proc. of the Euro-Par*, volume 2400 of *Lecture Notes in Computer Science (LNCS)*, pages 927–932, Paderborn, Germany, 2002. Springer-Verlag, Heidelberg.
82. B. Eckel. *Thinking in Java.* MindView, 2002.
83. A.E. Eiben and J.K. Van der Hauw. Solving 3-SAT with adaptive genetic algorithms. In *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 81–86. IEEE Press, 1997.
84. S.E. Eklund. Empirical studies of neighborhood shapes in the massively parallel diffusion model. In G. Bittencourt and G. Ramalho, editors, *Brazilian Symposium on Artificial Intelligence (SBIA)*, volume 2507 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 185–194. Springer-Verlag, Heidelberg, 2002.
85. L.J. Eshelman, K.E. Mathias, and J.D. Schaffer. Convergence controlled variation. In R. Belew and M. Vose, editors, *Foundations of Genetic Algorithms IV (FOGA)*, pages 203–224, San Mateo, CA, 1989. Morgan Kaufmann.
86. L.J. Eshelman and J.D. Schaffer. Real coded genetic algorithms and interval schemata. In L.D. Whitley, editor, *Foundations of Genetic Algorithms II (FOGA)*, pages 187–202, San Mateo, 1993. Morgan Kaufmann.
87. M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42-44:626–642, 1994.
88. M.J. Flynn. Very high speed computing systems. *Proc. IEEE*, 54:1901–1909, 1966.
89. D.B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60:139–144, 1988.
90. L.J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.
91. G. Folino, C. Pizzuti, and G. Spezzano. Combining cellular genetic algorithms and local search for solving satisfiability problems. In *Proc. of the IEEE International Conference on Tools with Artificial Intelligence*, pages 192–198, Taipei, Taiwan, 1998. IEEE Press.
92. G. Folino, C. Pizzuti, and G. Spezzano. Parallel hybrid method for SAT that couples genetic algorithms and local search. *IEEE Transactions on Evolutionary Computation*, 5(4):323–334, August 2001.

93. 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.
94. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Fransisco, 1999.
95. A.S. Fraser. Simulation of genetic systems by automatic digital computers II: Effects of linkage on rates under selection. *Australian Journal of Biological Sciences*, 10:492–499, 1957.
96. M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, 1979.
97. M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.
98. M. Giacobini, E. Alba, A. Tettamanzi, and M. Tomassini. Modelling selection intensity for toroidal cellular evolutionary algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 3102 of *Lecture Notes in Computer Science (LNCS)*, pages 1138–1149. Springer-Verlag, Heidelberg, 2004.
99. M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 955–966. Springer-Verlag, Heidelberg, 2003.
100. M. Giacobini, M. Tomassini, and A. Tettamanzi. Modelling selection intensity for linear cellular evolutionary algorithms. In P. Liardet et al., editor, *Proc. of the International Conference on Artificial Evolution*, volume 2936 of *Lecture Notes in Computer Science (LNCS)*, pages 345–356. Springer-Verlag, Heidelberg, 2003.
101. M. Giacobini, M. Tomassini, and A. Tettamanzi. Takeover time curves in random and small-world structured populations. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 1333–1340, Washington D.C. USA, June 25–29 2005. ACM Press.
102. M. Giacobini, M. Tomassini, A.G.B. Tettamanzi, and E. Alba. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505, October 2005.
103. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.
104. F.W. Glover and G.A. Kochenberger. *Handbook of Metaheuristics*. International Series in Operations Research Management Science. Kluwer Academic Publishers, 2003.
105. D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.
106. 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 I (FOGA)*, pages 69–93, San Mateo, CA, USA, 1991. Morgan Kaufmann.
107. D.E. Goldberg, K. Deb, and J. Horn. Massively multimodality, deception and genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 37–46. North-Holland, 1992.

108. D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 56–64. Morgan Kaufmann, 1993. San Mateo.
109. B.L. Golden, E.A. Wasil, J.P. Kelly, and I-M. Chao. *Fleet Management and Logistics*, chapter The Impact of Metaheuristics on Solving the Vehicle Routing Problem: algorithms, problem sets, and computational results, pages 33–56. Kluwer Academic Publishers, Boston, 1998.
110. V. Gordon, K. Mathias, and D. Whitley. Cellular genetic algorithms as function optimizers: Locality effects. In *ACM Symposium on Applied Computing (SAC)*, pages 237–241. ACM Press, 1994.
111. V. Gordon, R. Pirie, A. Wachter, and S. Sharp. Terrain-based genetic algorithm (TBGA): Modeling parameter space as terrain. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 229–235, 1999.
112. V. Gordon, D. Whitley, and A. Böhm. Dataflow parallelism in genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 533–542. North-Holland, 1992.
113. V.S. Gordon and J. Thein. Visualization tool for a terrain-based genetic algorithm. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 401–406. IEEE Press, 2004.
114. V.S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 177–183. Morgan Kaufmann, 1993.
115. M. Gorges-Schleuter. ASPARAGOS - an asynchronous parallel genetic optimization strategy. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 422–428. Morgan Kaufmann, 1989.
116. M. Gorges-Schleuter. Comparison of local mating strategies in massively parallel genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 553–562. North-Holland, 1992.
117. M. Gorges-Schleuter. Asparagos96 and the traveling salesman problem. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 171–174. IEEE Press, 1997.
118. M. Gorges-Schleuter. A comparative study of global and local selection in evolution strategies. In *Proc. of the International Conference on Parallel Problem Solving from Nature V (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science (LNCS)*, pages 367–377. Springer-Verlag, Heidelberg, 1998.
119. M. Gorges-Schleuter. An analysis of local selection in evolution strategies. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, volume 1, pages 847–854, San Francisco, CA, USA, 1999. Morgan Kaufmann.
120. J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(2):35–52, 2002.
121. J. Gottlieb and N. Voss. Representations, fitness functions and genetic operators for the satisfiability problem. In *Artificial Evolution*, Lecture Notes in Computer Science (LNCS), pages 55–68. Springer-Verlag, Heidelberg, 1998.
122. R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

123. P. Green. Phrap. http://www.phrap.org/phredphrapconsed.html.
124. A.O. Griewangk. Generalized descent of global optimization. *Journal of Optimization, Theory, and Applications*, 34:11–39, 1981.
125. C. Grimme and K. Schmitt. Inside a predator-prey model for multi-objective optimization: A second study. In M. Cattolico, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 707–714, Seattle, Washington, USA, July 8–12 2006. ACM Press.
126. F. Herrera, E. Herrera-Viedma, M. Lozano, and J.L. Verdegay. Fuzzy tools to improve genetic algorithms. In *Proc. Second European Congress on Intelligent Techniques and Soft Computing*, pages 1532–1539, 1994.
127. F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–62, April 2000.
128. F. Herrera, M. Lozano, and J.L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for the behavioral analysis. *Artificial Intelligence Reviews*, 12(4):265–319, 1998.
129. D. Hillis. Co-evolving parasites improve simulated evolution as an optimizing procedure. *Physica D*, 42:228–234, 1990.
130. F. Hoffmeister. *Applied Parallel and Distributed Optimization*, chapter Scalable Parallelism by Evolutionary Algorithms, pages 175–198. Springer-Verlag, Heidelberg, 1991.
131. L. Hogie, F. Guinand, and P. Bouvry. *The Madhoc Metropolitan Adhoc Network Simulator*. Université du Luxembourg and Université du Havre, France. Available at http://www-lih.univ-lehavre.fr/˜hogie/madhoc/.
132. L. Hogie, M. Seredynski, F. Guinand, and P. Bouvry. A bandwidth-efficient broadcasting protocol for mobile multi-hop ad hoc networks. In *International Conference on Networking (ICN)*, page 71. IEEE Press, 2006.
133. J.H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962.
134. J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
135. X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9(9):868–877, 1999.
136. S. Huband, L. Barone, R.L. While, and P. Hingston. A scalable multi-objective test problem toolkit. In C.A. Coello, A. Hernández, and E. Zitler, editors, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, pages 280–295, 2005.
137. T.S. Hussain. An introduction to evolutionary computation. Tutorial presentation. CITO Researcher Retreat, May 12-14, Hamilton, Ontario 1998.
138. H. Ishibuchi, T. Doi, and Y. Nojima. Effects of using two neighborhood structures in cellular genetic algorithms for function optimization. In T.P. Runarsson et al., editor, *Proc. of the International Conference on Parallel Problem Solving from Nature IX (PPSN-IX)*, volume 4193 of *Lecture Notes in Computer Science (LNCS)*, pages 949–958, Reykjavik, Iceland, September 2006. Springer-Verlag, Heidelberg.
139. S. Janson, E. Alba, B. Dorronsoro, and M. Middendorf. Hierarchical cellular genetic algorithm. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 3906 of *Lecture Notes in Computer Science (LNCS)*, pages 111–122, Budapest, Hungary, April 2006. Springer-Verlag, Heidelberg.

140. K.A. De Jong, M.A. Potter, and W.M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 338–345. Morgan Kaufmann, 1997.
141. H. Juille and J.B. Pollack. *Advances in Genetic Programming 2*, chapter Massively parallel genetic programming, pages 339–358. The MIT Press, MA, USA, 1996.
142. R.M. Karp. Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane. *Mathematics of Operations Research*, 2:209–224, 1977.
143. H.A. Kautz and B. Selman. Planning as satisfiability. In *European Conference on Artificial Intelligence*, pages 359–363, 1992.
144. S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In *Proc. of the ACM Computer Science Conference*, pages 66–73, Phoenix, Arizona, 1994. ACM Press.
145. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 4598:671–680, 1983.
146. M. Kirley. MEA: A metapopulation evolutionary algorithm for multi-objective optimisation problems. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 949–956. IEEE Press, 2001.
147. M. Kirley. A cellular genetic algorithm with disturbances: Optimisation using dynamic spatial interactions. *Journal of Heuristics*, 8:321–342, 2002.
148. M. Kirley and D.G. Green. An empirical investigation of optimization in dynamic environments using the cellular genetic algorithm. In D. Whitley et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 11–18, San Mateo, CA, 2000. Morgan Kaufmann.
149. M. Kirley, X. Li, and D.G. Green. Investigation of a cellular genetic algorithm that mimics landscape ecology. In X. Yao, editor, *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, volume 1585 of *Lecture Notes in Computer Science (LNCS)*, pages 90–97. Springer-Verlag, Heidelberg, 1999.
150. J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 98–105, Piscataway, NJ, 1999. IEEE Press.
151. J. Knowles and D. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2001.
152. U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:302–219, 1999.
153. J.R. Koza. Genetic programming. In J.G. Williams and A. Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998.
154. N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.
155. T. Krink and R. Thomsen. Self-organized criticality and mass extinction in evolutionary algorithms. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 1155–1161, Seoul, Korea, 2001. IEEE Press.

156. K.W.C. Ku. Enhance the baldwin effect by strengthening the correlation between genetic operators and learning methods. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 11071–11077, Vancouver, BC, Canada, July 16-21 2006. IEEE Press.
157. K.W.C. Ku, M.W. Mak, and W.C. Siu. Adding learning to cellular genetic algorithms for training recurrent neural networks. *IEEE Transactions on Neural Networks*, 10(2):239–252, March 1999.
158. P. Larrañaga and J.A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.
159. M. Laumanns, G. Rudolph, and H.P. Schwefel. A spatial predator-prey approach to multiobjective optimization: A preliminary study. In A.E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature V (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science (LNCS)*, pages 241–249, 1998.
160. C.-H. Lee, S.-H. Park, and J.-H. Kim. Topology and migration policy of fine-grained parallel evolutionary algorithms for numerical optimization. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 1, pages 70–76. IEEE Press, 2000.
161. J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.
162. F. Li, B. Golden, and E. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32:1165–1179, 2005.
163. L. Li and S. Khuri. A comparison of DNA fragment assembly algorithms. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335, 2004.
164. X. Li. A real-coded predator-prey genetic algorithm for multiobjective optimization. In C.M. Fonseca et al., editor, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pages 207–221. Springer-Verlag, Heidelberg, 2003.
165. X. Li and S. Sutherland. A cellular genetic algorithm simulating predator-prey interactions. In *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 416–421. Morgan Kaufmann, 2002.
166. H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, pages 61–68. ACM Press, 2000.
167. X. Llor and J.M. Garrell. Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In L. Spector et al., editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 461–468. Morgan Kaufmann, 2001.
168. H.R. Lourenco, O. Martin, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.
169. F. Luna, B. Dorronsoro, A.J. Nebro, E. Alba, and P. Bouvry. *Handbook on Mobile Ad Hoc and Pervasive Communications*, chapter Multiobjective Metaheuristics to Optimize the Broadcasting in MANETs. American Scientific Publishers, USA, 2007. To appear.

170. F. Luna, A.J. Nebro, B. Dorronsoro, E. Alba, P. Bouvry, and L. Hogie. Optimal broadcasting in metropolitan MANETs using multiobjective scatter search. In *European Workshop on Evolutionary Computation in Communication, Networks and Connected System (EvoCOMNET), en EvoWorkshops*, volume 3907 of *Lecture Notes in Computer Science (LNCS)*, pages 255–266, Budapest, Hungary, April 2006. Springer-Verlag, Heidelberg.
171. Z. Luo and H. Liu. Cellular genetic algorithms and local search for 3-SAT problem on graphic hardware. In G. Yen, editor, *Proc. of the IEEE Conference on Evolutionary Computation (CEC), IEEE World Congress on Computational Intelligence (WCCI)*, pages 10345–10349, Vancouver, BC, Canada, July 16-21 2006. IEEE Press.
172. G. Luque, E. Alba, and B. Dorronsoro. *Parallel Genetic Algorithms*, chapter 5, Parallel Metaheuristics: A New Class of Algorithms, pages 107–125. John Wiley & Sons, 2005.
173. F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.
174. J. Madera, E. Alba, and A. Ochoa. *Parallel Metaheuristics: A New Class of Algorithms*, chapter Parallel Estimation of Distribution Algorithms, pages 203–222. John Wiley & Sons, 2005.
175. T. Mahnig and H. Mühlenbein. Comparing the adaptive Boltzmann selection schedule SDS to truncation selection. In *II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization*, pages 121–128, La Habana, 1999.
176. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithm. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 428–433. Morgan Kaufmann, 1989.
177. T. Maruyama, T. Hirose, and A. Konagaya. A fine-grained parallel genetic algorithm for distributed parallel systems. In *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 184–190, San Francisco, CA, USA, 1993. Morgan Kaufmann.
178. T. Maruyama, A. Konagaya, and K. Konishi. An asynchronous fine-grained parallel genetic algorithm. In *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, Lecture Notes in Computer Science (LNCS), pages 563–572. North-Holland, 1992.
179. D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *National Conference on Artificial Intelligence*, pages 321–326, Providence, RI, 1997.
180. G. Mendel. *Versuche über Pflanzen-Hybriden*. Verhandlungen des Naturforschedes Vereines in Brünn 4, 1865.
181. Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, Heidelberg, third edition, 1996.
182. D.G. Mitchell, B. Selman, and H.J. Levesque. Hard and easy distributions for SAT problems. In P. Rosenbloom and P. Szolovits, editors, *Proc. of the Tenth National Conference on Artificial Intelligence*, pages 459–465, California, 1992. AAAI Press.
183. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.
184. P. Moscato. *Handbook of Applied Optimization*, chapter Memetic Algorithms. Oxford University Press, 2000.

185. H. Mühlenbein. Parallel genetic algorithms, population genetic and combinatorial optimization. In *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 416–421. Arlington, 1989.
186. H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–88, 1988.
187. H. Mühlenbein and G. Paab. From recombination of genes to the estimation of distributions I. Binary parameters. In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature IV (PPSN-IV)*, volume 1411 of *Lecture Notes in Computer Science (LNCS)*, pages 178–187. Springer-Verlag, Heidelberg, 1996.
188. 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.
189. H. Mühlenbein, M. Schomish, and J. Born. The parallel genetic algorithm as a function optimizer. *Parallel Computing*, 17:619–632, 1991.
190. T. Murata and M. Gen. Cellular genetic algorithm for multi-objective optimization. In *Proc. of the Fourth Asian Fuzzy System Symposium*, pages 538–542, 2002.
191. T. Nakashima, T. Ariyama, and H. Ishibuchi. Combining multiple cellular genetic algorithms for efficient search. In *Proc. of the Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, pages 712–716, 2002.
192. T. Nakashima, T. Ariyama, T. Yoshida, and H. Ishibuchi. Performance evaluation of combined cellular genetic algorithms for function optimization problems. In *Proc. of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 295–299, Kobe, Japan, July 16-20 2003. IEEE Press.
193. A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. A cellular genetic algorithm for multiobjective optimization. In D.A. Pelta and N. Krasnogor, editors, *Proceedings of the NICSO*, pages 25–36, Granada, Spain, 2006.
194. A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 2007. To appear.
195. A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. A study of strategies for neigborhood replacement and archive feedback in a multiobjective cellular genetic algorithm. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Proc. of the International Conference on Evolutionary Multi-criterion Optimization (EMO)*, volume 4403 of *Lecture Notes in Computer Science (LNCS)*, pages 126–140. Springer-Verlag, Heidelberg, 2007.
196. N. Nedjah, E. Alba, and L. de Macedo Mourelle. *Parallel Evolutionary Computations*. Studies in Computational Intelligence. Springer-Verlag, Heidelberg, 2006.
197. M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
198. S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. of the Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162, 1999.
199. Y.S. Ong and A.J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, April 2004.

200. I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41:421–451, 1993.
201. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
202. P.M. Pardalos and H.E. Romeijn. *Handbook of global optimization. Volume 2*. Kluwer Academic Publishers, 2002.
203. J.L. Payne and M.J. Eppstein. Emergent mating topologies in spatially structured genetic algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 207–214, Seattle, Washington, USA, 2006. ACM Press.
204. A. Pelc. *Handbook of Wireless Networks and Mobile Computing*, chapter Broadcasting In Wireless Networks, pages 509–528. John Wiley & Sons, 2002.
205. W. Peng and X.-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proc. of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 129–130. IEEE Press, 2000.
206. F.B. Pereira, J. Tavares, P. Machado, and E. Costa. GVR: a new representation for the vehicle routing problem. In *Irish Conference Proceedings on Artificial Intelligence and Cognitive Science (AICS)*, pages 95–102, Ireland, 2002. Springer-Verlag, Heidelberg.
207. C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002, May 2004.
208. T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter Jr. On the capacitated vehicle routing problem. *Mathematical Programming Series B*, 94:343–359, 2003.
209. I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., UK, 1965.
210. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
211. C.R. Reeves and J.E. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory*. Operations Research/Computer Science Series. Springer-Verlag, Heidelberg, 2003.
212. G. Reinelt. TSPLIB: A travelling salesman problem library. *ORSA Journal on Computing*, 3:376–384 URL: http://www.iwr.uni–heidelberg.de/groups/comopt/software/TSPLIB95/, 1991.
213. R. Reiter and A. Mackworth. A logical framework for depiction and image interpretation. *Applied Intelligence*, 41(3):123–155, 1989.
214. J.L. Ribeiro-Filho, C. Alippi, and P. Treleaven. *Parallel Genetic Algorithms: Theory and Applications*, chapter Genetic algorithm programming environments, pages 65–83. IOS Press, 1993.
215. P. Rickers, R. Thomsen, and T. Krink. Applying self-organized criticality to the diffusion model. In D. Whitley, editor, *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 325–330, Las Vegas, Nevada, USA, 2000.
216. G. Robertson. Parallel implementation of genetic algorithms in a classifier system. In *Proc. of the Second International Conference on Genetic Algorithms (ICGA)*, pages 140–147, 1987.

217. Y. Rochat and E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.
218. G. Rudolph. On takeover times in spatially structured populations: Array and ring. In K.K. Lai, O. Katai, M. Gen, and B. Lin, editors, *Proc. of the Second Asia-Pacific Conference on Genetic Algorithms and Applications (APGA)*, pages 144–151. Global-Link Publishing Company, 2000.
219. G. Rudolph and J. Sprave. A cellular genetic algorithm with self adjusting acceptance threshold. In *International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, pages 365–372, 1995.
220. 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, *Proc. of the International Conference on Parallel Problem Solving from Nature IV (PPSN-IV)*, volume 1141 of *Lecture Notes in Computer Science (LNCS)*, pages 236–244. Springer-Verlag, Heidelberg, 1996.
221. J. Sarma and K.A. De Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 181–186. Morgan Kaufmann, 1997.
222. J.D. Schaffer and L.J. Eshelman. On crossover as an evolutionary viable strategy. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 61–68. Morgan Kaufmann, 1991.
223. D. Schlierkamp-Voosen and H. Mühlenbein. Adaption of population sizes by competing subpopulations. In *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, pages 330–335, Piscataway, NY, 1996. IEEE Press.
224. B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51:123–143, 1999.
225. H.-P. Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*. PhD thesis, Technical University of Berlin, 1965.
226. H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, England, 1981.
227. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *National Conference on Artificial Intelligence*, pages 337–343, California, 1994. AAAI Press.
228. J. Setubal and J. Medianis. *Introduction to Computational Molecular Biology*, chapter 4 - Fragment Assembly of DNA, pages 105–139. University of Campinas, Brazil, 1997.
229. D. Simoncini, S. Verel, P. Collard, and M. Clergue. Anisotropic selection in cellular genetic algorithms. In *Proc. of the Genetic and Evolutionary Computation COnference (GECCO)*, pages 559–566, Seattle, Washington, USA, 2006. ACM Press.
230. M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Lecture Notes in Computer Science (LNCS). Springer-Verlag, Heidelberg, 1997.

231. J.E. Smith and F. Vavak. Replacement strategies in steady state genetic algorithms: static environments. In Banzhaf and Reeves, editors, *Foundations of Genetic Algorithms V (FOGA)*, pages 219–234. Morgan Kaufmann, 1998.
232. P. Spiessens and B. Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 279–286. Morgan Kaufmann, 1991.
233. J. Sprave. Linear neighborhood evolution strategies. In A.V. Sebald and L.J. Fogel, editors, *Proc. of the Annual Conference on Evolutionary Programming*, pages 42–51, River Edge, NJ, USA, 1994. World Scientific.
234. J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. In P.J Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proc. IEEE International Conference on Evolutionary Computation (CEC)*, volume 2. IEEE Press, 1999.
235. J. Stender. *Parallel Genetic Algorithms: Theory and Applications*. IOS Press, Amsterdam, The Netherlands, 1993.
236. D.R. Stinson. *An Introduction to the Design and Analysis of Algorithms*. The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 1985 (second edition, 1987).
237. I. Stojmenovic and J. Wu. *Mobile Ad Hoc Networking*, chapter Broadcasting and activity scheduling in ad hoc networks, pages 205–229. Wiley-IEEE Press, 2004.
238. R. Storn and K. Price. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.
239. P.D. Surry and N.J. Radcliffe. RPL2: A language and parallel framework for evolutionary computing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature III (PPSN-III)*, pages 628–637, Berlin, 1994. Springer-Verlag, Heidelberg.
240. E. Taillard. Parallel iterative search methods for vehicle-routing problems. *Networks*, 23(8):661–673, 1993.
241. E.-G. Talbi. *Parallel Combinatorial Optimization*. John Wiley & Sons, 2006.
242. E.-G. Talbi and P. Bessire. A parallel genetic algorithm for the graph partitioning problem. In *Proc. of the International Conference on Supercomputing*, pages 312–320. ACM Press, 1991.
243. R. Tanese. Distributed genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 434–439. Morgan Kaufmann, 1989.
244. R. Thomsen, P. Rickers, and T. Krink. A religion-based spatial model for evolutionary algorithms. In H.-P. Schwefel, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, and J.J. Merelo, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, Paris, France, 2000. Springer-Verlag, Heidelberg.
245. M. Tomassini. The parallel genetic cellular automata: Application to global function optimization. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Proc. of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391. Springer-Verlag, Heidelberg, 1993.
246. M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Natural Computing Series. Springer-Verlag, Heidelberg, 2005.

247. A. Töorn and Ž. Antanas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Heidelberg, Berlin, Germany, 1989.
248. P. Toth and D. Vigo. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2001.
249. P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.
250. S. Tsutsui and Y. Fujimoto. Forking genetic algorithm with blocking and shrinking modes. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 206–213, San Mateo, CA, 1993. Morgan Kaufmann.
251. S. Tsutsui, A. Ghosh, D. Corne, and Y. Fujimoto. A real coded genetic algorithm with an explorer and exploiter populations. In T. Bäck, editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 238–245. Morgan Kaufmann, 1997.
252. A. Van Breedam. *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle Related, Customer-related, and Time-related Constraints*. PhD thesis, University of Antwerp - RUCA, Belgium, 1994.
253. A. Van Breedam. A parametric analysis of heuristics for the vehicle routing problem with side-constraints. *European Journal of Operations Research*, 137:348–370, 2002.
254. D.A. Van Veldhuizen and G.B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, 1998.
255. H.-M. Voigt, H. Mühlenbein, and D. Cetković. Fuzzy recombination for the breeder genetic algorithm. In L. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms (ICGA)*, pages 104–111, 1995.
256. H. M. Voigt, I. Santibáñez-Koref, and J. Born. Hierarchically structured distributed genetic algorithms. In R. Männer and B. Manderick, editors, *Proc. of the International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, pages 155–164, Amsterdan, 1992. North-Holland.
257. K. Weinert, J. Mehnen, and G. Rudolph. Dynamic neighborhood structures in parallel evolution strategies. *Complex Systems*, 13(3):227–243, 2001.
258. P.M. White and C.C. Pettey. Double selection vs. single selection in diffusion model GAs. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, pages 174–180. Morgan Kaufmann, 1993.
259. D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, page 658, California, CA, USA, 1993. Morgan Kaufmann.
260. D. Whitley, R. Beveridge, C. Graves, and K. Mathias. Test driving three 1995 genetic algorithms: New test functions and geometric matching. *Journal of Heuristics*, 1:77–104, 1995.
261. D. Whitley, S. Rana, J. Dzubera, and K.E. Mathias. Evaluating evolutionary algorithms. *Applied Intelligence*, 85:245–276, 1997.
262. D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J.D. Schaffer, editor, *Proc. of the Third International Conference on Genetic Algorithms (ICGA)*, pages 133–140. Morgan Kaufmann, 1989.

263. B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 194–205. ACM Press, 2002.
264. S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific, Singapore, 1986.
265. D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.
266. S. Wright. Isolation by distance. *Genetics*, 28:114–138, 1943.
267. J. Wu and W. Lou. Forward-node-set-based broadcast in clustered mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 3(2):155–173, 2003.
268. F. Xhafa. A cellular memetic algorithm for resource allocation in grid systems. In E.-G. Talbi and L. Jourdan, editors, *Proceedings of the META 2006*, 2006. CD edition.
269. F. Xhafa, E. Alba, and B. Dorronsoro. Efficient batch job scheduling in grids using cellular memetic algorithms. In *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Simposium (IPDPS) Workshop*, pages 1–8. IEEE Press, 2007.
270. E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *IEEE Transactions on Evolutionary Computation*, 8(2):173–195, 2000.
271. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), 2001.
272. E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 23(4):257–271, 1999.

Index

Early Titles in
OPERATIONS RESEARCH/COMPUTER SCIENCE INTERFACES

Greenberg /*A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*
Greenberg / *Modeling by Object-Driven Linear Elemental Relations: A Users Guide for MODLER*
Brown & Scherer / *Intelligent Scheduling Systems*
Nash & Sofer / *The Impact of Emerging Technologies on Computer Science & Operations Research*
Barth / *Logic-Based 0-1 Constraint Programming*
Jones / *Visualization and Optimization*
Barr, Helgason & Kennington / *Interfaces in Computer Science & Operations Research: Advances in Metaheuristics, Optimization, & Stochastic Modeling Technologies*
Ellacott, Mason & Anderson / *Mathematics of Neural Networks: Models, Algorithms & Applications*
Woodruff / *Advances in Computational & Stochastic Optimization, Logic Programming, and Heuristic Search*
Klein / *Scheduling of Resource-Constrained Projects*