An Optimization Approach for Software Test Data Generation: Applications of Estimation of Distribution Algorithms and Scatter Search

Dissertation

Ramón Sagarna
Department of Computer Science and Artificial Intelligence
University of the Basque Country
Supervised by: J. A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country in partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, January 2007

Acknowledgements

The present dissertation would have never been achieved without the support of a variety of people.

I am particularly indebted to Jose Antonio Lozano, my thesis supervisor. Undoubtedly, his wise, while at the same time friendly guidance throughout these years has made me grow a lot. Jose Antonio, thanks for your invaluable help and patience.

I am also grateful to Pedro Larrañaga, Iñaki Inza, Alex Mendiburu, Endika Bengoetxea and the rest of my colleagues at the Intelligent Systems Group. Their encouragement and advice have been decisive as well. I would like to make special mention of my lab pals: Ruben Armañanzas, Borja Calvo, Yosu Galdiano, Dinora Morales, Aritz Pérez, Juan Diego Rodríguez, Guzmán Santafé, Roberto Santana and John Kennedy. Working with them has been extremely easy because of the joyful working environment we have shared, both inside and outside the lab. Those that have been with us temporarily have also contributed beneficiously: Robin Höns, Siddartha Shakya, Frederick Vincent and others. Thanks Robin for the good time in Munich.

My most sincere gratitude to Walter Gutjahr and Immanuel Bomze, who were my hosts in the Department of Statistics and Decision Support Systems at the University of Vienna. The enthusiastics long discussions with Walter have been enlightening, and the keen talks with Immanuel very enriching. I also have to give special mention the rest of the people from that department and the friends I left in Vienna. The experience there is somehow reflected in this work.

Acknowledgements to the Department of Computer Science and Artificial Intelligence from the University of the Basque Country, and to the Basque Government for the financial contribution.

Finally, I would like to thank my family and friends. “Ama” and “Aita”, I am in debt to you for the continuous support and understanding, which has been essential for concluding the thesis.

This dissertation is dedicated to you all.

Contents

List of Figures

List of Tables

1 Introduction

The recurring concept in optimization is to select the best alternative among a number of possible results or affairs. Mathematically, optimization is the minimization or maximization of a function subject to constraints on its variables. Therefore, three basic elements may be recognized from this description: the function, the set of variables and their constraints, and a strategy for finding the function extrema.

A first step in the optimization process is the identification of the function, variables and constraints for a given practical problem. This task is sometimes referred to in the literature as *modeling* [158]. The set of variables and constraints represents the features that distinguish the results from one another, that is, they define the possible problem solutions. The whole set of such solutions is usually called the *search space*. On the other hand, the function provides with a quantitative criterion of merit, according to which one solution can be classified as better or worse than others. This is known as the *objective function*, as it depends on the goal to be attained. Construction of an appropriate model is a relevant, and sometimes the hardest, issue. If the model is too simplistic, it may not give useful insights into the problem, while if it is too complex, it could become extremely difficult to solve.

The second topic in optimization lies in the choice or design of a suitable strategy for proceeding. There is no universal optimization technique. Rather, there are numerous methods, each of which is tailored to a particular type of problem. Most classical methods are founded on theoretical concepts regarding the necessary and sufficient conditions for the existence of extrema. However, usually a long way exists from the establishment of such conditions to their determination. It often remains intrincate enough, if not impossible, to find the optimum for many problems. Even *exact* methods, which ensure reaching the optimum, have well-known computational limitations which turn them into an unfeasible option, e.g. for NP-hard problems [83]. In tackling such problems, *metaheuristic* techniques become one of the most successful alternatives since, although optimality is not often guaranteed, a high quality solution at a reasonable computational cost is usually obtained.

The large number of existing metaheuristics makes it difficult to classify them accurately. One of the strategies that has grown into a popular field is Evolutionary Algorithms (EAs) [7]. EAs are a family of probability based methods that take a model for the natural evolution of species, formulated by Darwin [50], as a source of inspiration. More precisely,

the search for the optimum proceeds by maintaining a *population* of solutions that *evolves* from one generation to the next. The evolution consists of selecting a set of solutions from the population and applying to some subsets of it *recombination* operators that create new solutions. A huge number of methods conforming to this framework have been developed. Therefore, the choice of the appropriate alternative for a particular application results in an important matter, as it may determine whether the problem is solved efficiently or, even, if the optimum is found at all.

Two modern emerging EAs techniques are Estimation of Distribution Algorithms (EDAs) [130] and Scatter Search (SS) [125]. The term EDAs alludes to a group of algorithms which, instead of using the typical recombination operators from EAs, sample a probability distribution previously built from the set of selected solutions. Indeed, this distribution is responsible for one of the main characteristics of these algorithms, that is, the explicit description of the relationships between the problem variables. Alternatively, SS is a methodology based on the support of a low cardinality set of solutions which is updated with new solutions obtained from the combination of the members of the set. Probably the most genuine feature of SS is that it emphasizes the use of systematic rules during the process, though stochasticity can remain to some extent.

A significant aspect in the study of optimization methods is their application to real-world problems. This is appealing not just to demonstrate their usefulness, but also to uncover limitations that only arise in realistic situations. Optimization and, more specifically, EAs have been applied to problems from a wide range of areas such as economics, manufacturing, physical systems, biology or operations research, just to name a few. A relatively unexplored discipline is, however, software testing.

Testing is the means used in practice to verify the correctness of software produced. Considering the crucial role of software nowadays, it is not difficult to imagine the significance of testing. In fact, it represents a major issue for quality assurance and it usually accounts for 50% of project resources [17]. A huge amount of these resources is dedicated to the generation of the input cases to be applied to the program under test. This task is not trivial, as input cases must conform to the test type and its requirements. Since most organizations perform this step manually, the automatic generation of test data is worthwhile and has turned into one of the most challenging problems in the area. A common strategy for facing this task consists of creating test inputs that fulfill an adequacy criterion based on the program structure. That is, adequacy criteria come defined by the entities revealed by the program source code. For example, entities such as the branches the flow of control can take from a conditional statement define the *branch coverage* criterion, i.e. every program branch must be exercised.

In the last few years, a number of approaches under the name of Search Based Software Test Data Generation (SBSTDG) have been proposed, offering interesting results [143]. SBSTDG deals with the test data generation as a search for the appropriate inputs by

formulating an optimization problem during the process. This problem may then be solved using metaheuristic search techniques.

This dissertation is devoted to the treatment of the test data generation problem from an optimization point of view. More precisely, the three major components of an optimization problem are studied in the context of branch coverage, which is a mandatory criterion nowadays.

Concerning problem modeling, both the search space and the objective function are investigated. In the literature related to this problem, little attention has been paid to the search space topic. The present work aims at revealing its significance for obtaining improved results in terms of efficiency and efficacy. For this, in order to select a promising region, a strategy that dynamically transforms the search space during the process is developed. By contrast, a much more intensive effort has been deserved in the bibliography to the objective function. It is worth discussing then different functions previously proposed and to study them in the present scope with the purpose of uncovering their influence.

The current work emphasizes the optimization technique topic, focusing on the application of EDAs and SS. The main objective regarding this subject is twofold: on the one hand, to show that leading edge metaheuristics are able to perform successfully in this problem and contribute new alternatives for its solution, and on the other hand, to ascertain whether EDAs and SS become practical methods in a demanding real world problem.

1.1 Outlook of the Dissertation

This dissertation is composed of seven chapters. Chapter 2 presents the optimization methods studied throughout this work, namely, EDAs and SS. The general optimization problem and the concept of metaheuristic are firstly introduced. EAs are then described by giving a general framework to which every algorithm roughly conforms. Though the origins of EDAs are not clearly stated, Genetic Algorithms seem to be one of their sources; a little more attention is devoted to them. Finally, SS and, more extensively, EDAs are overviewed. For the latter, existings methods are classified in three groups: those where the probabilistic model assumes problem variables are conditionally independent, algorithms with first order dependence probability distributions, and EDAs where the model makes no restriction on the dependencies between variables. Apropos SS, apart from the general methodology, advanced designs as well as applications are included.

The purpose of Chapter 3 is to explain the problem faced in the dissertation. The need for software testing is motivated by describing its relevance with regard to quality assurance. Discussion concentrates on the generation of test inputs. The basic strategies

for accomplishing this task are explained, pointing out their limitations. The most common strategy consists of fulfilling a code coverage criterion. This concept, together with its complexity, are introduced. In the last part of the chapter, SBSTDG is dealt with. There, a general schema followed by many approaches is described. Additionally, two designs for the objective function are presented and the optimization methods used in the literature reviewed.

The following three chapters study the topics involving the main novelty of the dissertation.

In Chapter 4, the test data generation is formulated from an optimization perspective. The application of EDAs to the general scheme from SBSTDG is then described in detail. Several EDAs are chosen for their evaluation through extensive experimentation.

Chapter 5 studies the application of SS. Again, the general scheme is employed as the basis for the approach. The role of the improvement method in the SS algorithm is analyzed by exposing different alternatives for its usage. Moreover, an EDA-SS combination is proposed in order to take advantage of the benefits of both approaches.

Chapter 6 concerns the two other elements of the optimization: the objective function and the search space. Two functions previously described are discussed and compared empirically to check their adequacy. The bulk of the chapter, however, tackles the search space issue. A strategy for the selection of a promising search region is widely described and experimentally evaluated. Then, diverse analysis of the results are included to validate its performance and obtain conclusions.

Finally, Chapter 7 lists the main contributions and conclusions of this work. Future lines of research are suggested as well.

2 Modern Evolutionary Optimization Techniques

In the past years, a significant research effort has been devoted to the study and development of optimization methods and, more specifically, of metaheuristics. As a result of this work, a number of methods are emerging which contribute new ideas in the field and improve the results of more classical alternatives in certain problems. Two representatives of such novel methods are Estimation of Distribution Algorithms (EDAs) and Scatter Search (SS). The former complies with a research line where optimization is based on probabilistic models, while the latter emphasizes a more systematic approach.

The aim of this chapter is to serve as an introduction to such methods. Firstly, the optimization problem, in general, and metaheuristics, in particular, are presented. EDAs and SS are typically included under the framework of Evolutionary Algorithms. The very basics of this framework are described then. Next, both EDAs and SS are overviewed. The chapter ends by pointing some other optimization methods which are deserving of the interest of the community as well.

2.1 Introduction to Metaheuristics

The classical objective of optimization is to find variables values leading to an extremum of a function. More formally, the general problem may be stated as follows: given a function $f : \Omega \rightarrow \mathbb{R}$, find $\boldsymbol{x}^* \in \Omega$ such that

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x}).$$

Function f is the *objective function* and the set $\Omega \subseteq \mathbb{R}^n$ is called the *feasible region*, though the term *search space* is usually employed as well. Additionally, Ω may come defined by a number of restrictions, formulated as functions on the problem variables, that is, $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, g_i(\boldsymbol{x}) \geq 0,\, g_i : \mathbb{R}^n \rightarrow \mathbb{R},\, i \in \{1, ..., m\}\}$. If $\Omega = \mathbb{R}^n$, the problem is called *unconstrained*, otherwise it is *constrained*.

Notice that according to the description above minimization is sought. This by no means causes a loss in generality, since

$$\max_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x}) = -\min_{\boldsymbol{x} \in \Omega}(-f(\boldsymbol{x})).$$

Moreover, f takes values in $\mathbb{R}$, even so it could be widened to $f : \Omega \rightarrow \mathbb{R}^k$; if $k > 1$, a *multiobjective optimization* problem is being posed [43]. Many challenging real-world problems involve, by nature, the attainment of multiple objectives, however, in the present work, we will restrict ourselves to a single objective, i.e. $k = 1$. Different levels of knowledge about the mathematical properties of f are possible. In the case of no knowledge at all, the problem is named *black-box optimization*.

Regarding the topology of f, different types of minima can be distinguished. A minimum is called *local* if no smaller function value exists in the surroundings of the corresponding point in Ω. The deepest of the local minima is known as the *global* one. More precisely, given a point $\boldsymbol{x_*} \in \Omega$, $f(\boldsymbol{x_*})$ is a local minimum if an $\varepsilon > 0$ exists such that $\forall \boldsymbol{x} \in \Omega \;\; \|\boldsymbol{x} - \boldsymbol{x_*}\| < \varepsilon \Rightarrow f(\boldsymbol{x_*}) \leq f(\boldsymbol{x})$. In the case of a global minimum, this holds for every ε, that is, $\forall \boldsymbol{x} \in \Omega \;\; f(\boldsymbol{x_*}) \leq f(\boldsymbol{x})$. It can be noted that the concept of local minimum depends, to a large extent, on the metric. In the case of continuous problem variables, a common practice is to adopt the Euclidean distance. If variables are discrete, however, a mapping $N : \Omega \rightarrow 2^{\Omega}$ is defined on the basis of a metric; such a mapping is referred to as a *neighborhood*.

An objective function that only shows one local minimum is named *unimodal*, otherwise it is called *multimodal*. Obviously, we are interested in finding a global optimum of f, therefore, in the previous description of an optimization problem, the solution point $\boldsymbol{x^*}$ alludes to a global minimum. From now on, terms optimum and minimum will be related with globality, unless otherwise specified where confusion might arise.

Inherent to an optimization problem is the need for a suitable solution strategy. No general-purpose effective method has been found, so the field is covered by procedures that limit their application to specific problem cases each. A rough classification of these methods can be drawn by considering the problem features. Thus, for instance, we may discern between strategies for *static* and *dynamic* optimization (extrema are stationary or of time-varying nature), *parameter* and *functional* optimization (problem variables are scalars or functions), or constrained and unconstrained optimization.

One other possible distinction is between *analytic* and *numerical* optimization methods [222]. Analytic (also known as *indirect*) procedures are based on the investigation of the particular properties of f at the extrema points. For this, classical theoretical concepts regarding the necessary and sufficient conditions for the existence of minima and maxima are employed, resulting in systems of equations that a solution must satisfy. However, difficulty, even possibility, of determination of this solution heavily depends on particular problem conditions; so, aspects like continuity or differentiability of f, whether nonlinear equations are involved, or the existence of constraints, strongly restrict the application of such strategies. On the other hand, numerical (or *direct*) methods are more widely spread than indirect. Direct methods consist of approaching the solution iteratively, attempting to improve the value of f at each step. Not achieving this improvement

causes a trial and error process which, in the uttermost case, leads to an exhaustive exploration of the search space. Indeed, strategies that ensure the achievement of the optimum, commonly known as *complete* or *exact*, are based on the examination of a large proportion of the search space. Unfortunately, limitations arise when their computation is addressed. Computational complexity associated to a procedure grows with the size of the search space; in case of an exponential growth, the problem is deemed *intractable*. Furthermore, a huge number of problems from diverse areas such as economics, biology or operations research belong to the NP-hard category [83], which implies no algorithm that attains the optimum in a polynomial time complexity scale is known. In consequence, these problems are considered to be intractable.

Hence, we arrive at a lack of feasible solution strategies for a significant number of problems. The unavoidable question is: how can they be tackled? An alternative is to approximate the optimum by means of *heuristics* [196]. A heuristic is a rule of thumb that gives guidance in the solution of a problem. Although optimality is not guaranteed, a high quality solution at a reasonable computational cost is usually achieved. This efficiency is very appreciated when facing complicated real-world problems and constitutes the clue for the extended application of these techniques.

Many heuristic methods consist of a search process over the feasible region [178]. Such heuristic search procedures can be further divided into *deterministic* and *nondeterministic* algorithms. In the formers, deterministic rules are used at each step of the process, that is, given a problem, two executions of the algorithm under identical conditions result in the same solution. By contrast, in nondeterministic approaches, several options are feasible at some decision points during the search. It is common to resort to stochastic rules at these points and, accordingly, different solutions might be attained in two runs of the same algorithm, given a problem and identical execution conditions. Examples of deterministic and nondeterministic approaches are, respectively, *coordinate hill climbing* [222] and random directions algorithms [261].

Taking the type of the extrema into account, search heuristics may also be classified as *local* or *global* methods. Local methods operate in the surroundings of a solution point at each step of the search, until an optimum is found[1]. If f is multimodal then a local optimum, different to the global, is often reached [261]. On the other hand, global methods aim at covering the search space to some extent, with the purpose of obtaining the global optimum. Respective instances of local and global algorithms comprise the *best first* procedure [178] and *grid search* strategies [222]. It is important to remark that, in many cases, these two types of methods are combined in order to build other global procedures, e.g. *multistart* algorithms [219] typically consist of multiple applications of a local search heuristic departing from different initial points each.

[1] Other works from the literature [32] simply refer to local methods as those employing a neighborhood. Notice that a stronger description is used here [7; 222; 261].

An enormous effort has been devoted in the past years to the study and development of heuristic methods. One of the main objectives of this work has been the improvement of the traditional heuristic algorithms, resulting in increasingly advanced designs. These have been included under the relatively recent term of *metaheuristics* [32; 86; 196]. Glover used this term for the first time to describe procedures consisting of "... a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality." [87]. Nonetheless, in practice, metaheuristics involve sophisticated as well as modern approaches [196]. The similar term of *hiper-heuristics* can also be found in the literature [201], though its meaning is clearly different, as it alludes to methods which seek through a search space of heuristic algorithms. A few well known examples of search metaheuristics are Simulated Annealing [120], Tabu Search [87] and Evolutionary Algorithms [7].

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [7; 8; 9; 66] is the term used to group a number of stochastic (nondeterministic) global search metaheuristic techniques. All these techniques share as a source of inspiration the theory of natural evolution of species enunciated by Darwin [50]. According to Darwin's model, the evolution of a population of individuals in an environment with limited resources is based on two cornerstones: natural selection and phenotypic variations. Natural selection favors reproduction of the best adapted members in the population, allowing their genetic information to spread through their offspring. Phenotypic variations given by genes recombination during reproduction and, occasionally, by small mutations of a gene, produce new individuals in the population.

EAs come motivated by the interpretation of adaptation as a successive progress of improvement of structures in order to attain a better performance in their environment. Natural evolution can be then observed as an optimization process which is worthy of imitation. This was the idea followed by three approaches developed separately during the 60s. In the USA, Holland introduced Genetic Algorithms [107], while Fogel, Owens and Walsh invented Evolutionary Programming [73]. Meanwhile, in Germany, Rechenberg and Schwefel implemented Evolution Strategies [195; 222]. Such techniques have been considered the mainstreams of the field since their development. EAs, however, comprise many more methods than these. Due to their extended use and their connection with the scope of this thesis, Genetic Algorithms will be briefly introduced later on. For a detailed explanation of these approaches and EAs in general, the reader is referred to [8; 9; 66; 73; 88; 222].

2.2.1 The General Framework

As a consequence of the evolutionary metaphor, much of the biological terminology has been transferred to the field of EAs. For instance, a candidate solution point for the problem at hand is represented by an *individual*, which is in turn composed of a set of *genes* or variables. The term *population* alludes to a set of individuals, that is, a set of representations of the candidate solutions. The objective function is referred to as the *fitness function* and, accordingly, the function value of an individual is called its *fitness*.

An EA lies on the basis formed by three stochastic operators, namely, *selection*, *recombination* and *mutation*. Recombination and mutation are not simultaneously included in all designs, though one of them is always present, e.g. basic Evolution Strategies only involve mutation. Many techniques, however, contain both operators.

The search for the optimum point consists basically of an iterative process departing from a population of individuals. At each round or *generation*, the selection operator is applied to choose a set of parent individuals from the current population. For this, fitter individuals are assigned a higher probability of being chosen. Recombination merges the genes of several parents (commonly two) into offspring individuals. The choice of the genes to be combined as well as the manner of combination are determined probabilistically. Even the application of the operator depends on a probability value. Mutation performs random transformations on the genes of one individual. Similarly to recombination, each of these transformations occurs with a certain probability. Finally, new individuals compete with the old ones for a place in the next generation. Figure 2.1 shows the pseudocode of a general framework to which every EA conforms to some extent. In the pseudocode, D_l represents the population of the l-th generation; analogously, D_l^{Sel}, D_l^{Rec} and D_l^{New} denote the set of selected individuals, the offspring after recombination and the new individuals after mutation, respectively.

$D_0 \leftarrow$ Obtain initial population
Evaluate each individual from D_0
Repeat for $l = 1, 2, ...,$ until stopping criterion is met
  $D_{l-1}^{Sel} \leftarrow$ Select individuals from D_{l-1}
  $D_{l-1}^{Rec} \leftarrow$ Recombine individuals from D_{l-1}^{Sel}
  $D_{l-1}^{New} \leftarrow$ Mutate individuals from D_{l-1}^{Rec}
  Evaluate each individual from D_{l-1}^{New}
  $D_l \leftarrow$ Build next population with individuals from D_{l-1} and D_{l-1}^{New}

Figure 2.1: General framework for EAs.

From an optimization point of view, EAs are based on two assumptions. On the one hand, each solution point is a container of knowledge about the features of the objective function. On the other hand, when solutions are combined, their knowledge is trans-

mitted to the resulting solutions. Since fitter individuals have higher chances of being selected, it is expected that the parents encode suitable features. As a result of their combination, better individuals could then be obtained, and the population might eventually evolve towards promising areas of the search space.

At this point, it is important to notice the significance of the encoding scheme used by individuals to represent candidate solutions. This scheme can be described as a mapping $h : \Omega \rightarrow \Omega_I$, where Ω_I is the set of individuals. The search proceeds in Ω_I, so the objective function of the problem actually being tackled is $f_I : \Omega_I \rightarrow \mathbb{R}$, instead of $f : \Omega \rightarrow \mathbb{R}$. In some cases, an individual is a solution point (h is the identity function) and, hence, $f_I = f$, while, in others, more elaborated mappings are required. Therefore, both h and f_I must be carefully defined in order to preserve the properties of the originals search space and objective function. A convenient approach, though not always feasible, is to make h bijective, so that $f_I = f \circ h^{-1}$.

The framework in Figure 2.1 reveals the remaining elements that characterize EAs. While some of these components are often specified following general rules, others tend to be tailored to the particular technique at hand. Thus, the initial population can be obtained independently from the algorithm through several alternative procedures, e.g. at random. By contrast, design of the evolutionary operators is usually influenced by the method. Below, a few operators are described in detail for the case of Genetic Algorithms. Apropos the construction of the next population, relatively simple rules, like choosing the individuals with highest fitness, are popular. It is worth noting a practice called *elitism*, which consists of preserving for the next population the best individual from the current one. The use of elitism is widely spread as it has shown good results in many problems as well as necessary properties for theoretical convergence to the global optimum [136; 204]. For the termination condition, specific strategies exist for a few EAs [5], though the usual approach is to resort to general criteria. For instance, if the optimal value is known, an obvious halting condition is the attainment of such a fitness. Further basic criteria are, among others, reaching a maximum number of fitness evaluations or computational resources, no improvement of the mean fitness of the population in succesive generations, or the convergence of the population to the same individual. In order to fully define an EA, parameters that come together with these elements should also be specified. Parameter values that need to be set in most techniques are the population size, the number of individuals selected, and probabilities of recombination and mutation. Of course, additional parameters may arise from the particular details of each component.

When compared with classical optimization methods, EAs have a number of advantages which make them an interesting option. First, EAs may be applied to a wide variety of scenarios. They are able to deal with functions where no derivatives exist, multimodality, discontinuities, constraints, or with noisy functions. Even problems not completely defined or black-box optimization can be tackled. Second, EAs do not make any assumption about the search space. Third, the effort to adapt an EA to a new problem is

relatively low. Finally, some EAs can run interactively, i.e. it is possible to change the parameter values during execution.

EAs have also drawbacks. First, there is no guarantee of finding the global optimum and, in most cases, no reliable stopping criteria are known. Second, EAs are complex systems that make their theoretical analysis fairly intrincate, leading to a lack of enough theoretical basis in the field. Moreover, comparison between different EAs is difficult unless experimentally. Third, they are often computationally expensive. Fourth, it is not possible to know how far the solution obtained is from the global optimum. Finally, perhaps their worst characteristic is a strong dependency on the set of parameters, which usually has to be experimentally tuned for the problem at hand. In fact, in some algorithms, this tuning itself becomes an optimization problem [93]. Exceptionally, the tuning process can be avoided by Evolution Strategies and other self-adaptive EAs [65]. To summarize, it is important to think that EAs are not a set of techniques ready to be applied, but a set of mechanisms to modify and tailor to the particular problem.

Nonetheless, the high quality results obtained in many problems have caused an exponential growth of the field. The literature is plenty of successful applications to real-world problems [51] as well as their abstract forms, like, for example, the Traveling Salesman Problem [7]. These encouraging results led a number of works from the 80s to conclusions on the superiority of EAs with regard to other techniques [88]. Still, the No Free Lunch Theorem [252] showed later on that, averaging over the space of possible problems, all black-box algorithms exhibit the same performance. Since then the perspective of researchers has completely changed. Instead of seeking the best overall method, current efforts are addressed towards the identification of the suitable methods for a given problem, or the study of the problems where a method performs well. Undoubtedly, much of this work has been devoted to the, probably, most popular EA: the Genetic Algorithm.

2.2.2 Genetic Algorithms

Origins of Genetic Algorithms (GAs) lie in the work by Holland [107], although their popularity is mainly due to Goldberg [88]. The pseudocode in Figure 2.1 can be straightly applied to GAs, since both recombination and mutation operators are contained in the algorithm. The Simple Genetic Algorithm (SGA) is the most elementary implementation of this method. Basic representation issues, together with the selection, recombination and mutation operators of the SGA, are described next. Additionally, a few advanced designs are included. Excellent in-depth discussions of GAs can be found in the literature; for instance, the reader may consult the book by Goldberg [88], or the more recent one by Vose [243].

Representation of a solution

An individual in the SGA encodes a solution point from the original problem as a binary string of n genes, i.e. an individual can be denoted as $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, with $\boldsymbol{x} \in \{0,1\}^n$. This encoding implies a mapping function must be defined to transform original solutions into 0-1 strings. For instance, an integer variable could be mapped to a binary string following a sign-magnitude representation [244].

Another possibility is to encode each solution in a natural, non binary, way, resulting in strings of integers or reals. A disadvantage of these encodings is that most of the evolutionary operators are built with regard to the SGA, and they might not be used with non 0-1 strings. More precisely, many of the operators can be applied to integer encodings, but not to reals. In these cases, specific operators usually need to be designed [105].

Selection

The purpose of selection is to push the search towards high quality areas. In the SGA, the number of individuals to be selected is a parameter of the algorithm and the operator is known as *proportional-based selection*. Assuming a population size N and an objective function f to maximize, the probability of choosing an individual $\boldsymbol{x_i}$ with this operator is

$$p(\boldsymbol{x_i}) = \frac{f(\boldsymbol{x_i})}{\sum_{j=1}^{N} f(\boldsymbol{x_j})} \tag{2.1}$$

A well known drawback of proportional-based selection is that it is not invariant under translation nor under a change in scale of the objective function. Therefore, more advanced strategies are used in practice. For example, in *linear rank-based selection* [10], previous drawbacks vanish, since a rank of the fitness values in the population is employed to elicite the probability of selection for an individual. Let η^+ denote the expected number of times the best individual $\boldsymbol{x_{1:N}}$ is selected, i.e. $\eta^+ = N \cdot p(\boldsymbol{x_{1:N}})$, and η^- the minimum expected value assigned to $\boldsymbol{x_{N:N}}$, i.e. $\eta^- = N \cdot p(\boldsymbol{x_{N:N}})$, then the probability assigned to individual $\boldsymbol{x_{i:N}}$ is elicited following a linear mapping, that is,

$$p(\boldsymbol{x_{i:N}}) = \frac{1}{N} \cdot \left(\eta^+ - (\eta^+ - \eta^-) \cdot \frac{i-1}{N-1} \right) \tag{2.2}$$

and the constraints $\sum_{i=1}^{N} p(\boldsymbol{x_{i:N}}) = 1$ and $p(\boldsymbol{x_{i:N}}) \geq 0 \; \forall i \in \{1, 2, ..., N\}$ imply $1 \leq \eta^+ \leq 2$ and $\eta^- = 2 - \eta^+$. As it can be noticed, the implementation of this operator involves a new parameter η^+.

Recombination

According to the EAs hypothesis, recombination is the way to spread the information of individuals. In the SGA, this operator is applied with a certain probability, which is given as a parameter. The probability value tends to be high (nearly 1) in order to facilitate the exploration of the search space. Two parents $(x_1^1, x_2^1, ..., x_n^1)$ and $(x_1^2, x_2^2, ..., x_n^2)$ are mixed and two children are obtained. The procedure consists of choosing an index k from a uniform distribution over $\{1, 2, ..., n-1\}$. Then, the first child is composed of genes $x_1^1, ..., x_{k-1}^1$ and $x_k^2, ..., x_n^2$, while the second is created with genes $x_1^2, ..., x_{k-1}^2$ and $x_k^1, ..., x_n^1$.

The underlying idea behind this cut point based operator is that neighbor genes contain the useful knowledge to obtain improved individuals. However, even if this is true, there is no apparent reason to assume that the neighborhood is given by adjacent variables in the string. Hence, several approaches have proposed to extend the number of cut points to two or more. The general situation is the *uniform recombination* operator [235], where for each gene, one of the parents is chosen randomly and independently of the rest of the variables.

Other recombination operators have investigated the possibility of merging more than two parents. In the Bit-based Simulated Crossover [236], the value of each gene is obtained from the value of the same variable in an individual chosen from the whole population. The selection of the individual consists of sampling from a probability distribution that depends on the fitness value of the individuals.

Mutation

The role of mutation is to inject diversity in the population, enhancing the exploration capabilities of the algorithm. In the SGA, the value of each gene is flipped with a probability value which is a parameter of the system. In contrast to recombination, this value is kept low (a rule of thumb is to use $1/n$ [148]) to avoid the excessive disruption of the effect of the recombination operator.

2.2.3 Advanced Designs

Intensive research has been committed to EAs in the past years. To a great extent, these efforts have concentrated on the alleviation of the drawbacks previously introduced or the improvement of the techniques performance. It is worth to remark then a number of approaches in the literature making use of advanced designs.

The hybridization of EAs with other techniques [22] is an active working line supported by the impressive results obtained in practice, e.g. in graph coloring problems [79]. One

other appealing area is based on the study of strategies for the self-adaptation of the parameters values of the EA [6; 65]. The purpose of these developments is to make the algorithm less dependant on the parameters by allowing the evolutionary process to change their values during the search. A more spread approach is the generalization of this idea, that is, designing EAs where a number of basic components are self-adaptive [65]. In this context, the bulk of the works concentrate on the employment of variable length representations of a solution [122; 250]. Finally, much of the research efforts are concerned with the parallelization of EAs [34; 203]. This field is not only interesting from an efficiency point of view, but also from a methodological one, since some parallel methods imply a different behavior from the classical EA and constitute themselves a new domain, e.g. island models GAs [34].

Of course, these approaches are just a few instances from the body of extensions and innovative ideas concerning EAs. In fact, such is the level of sophistication achieved by many recent developments that they do not fit exactly into the framework in Figure 2.1. The notion of EA is becoming increasingly blurred, favouring the inclusion of several techniques that follow some evolutionary concepts under its umbrella. Next, we review two leading edge techniques amongst these, namely, Estimation of Distribution Algorithms [130] and Scatter Search [125].

2.3 Estimation of Distribution Algorithms

In the last decade, GAs have been widely used to solve different problems, improving in many cases the results obtained by other algorithms. However, as it was pointed out in the previous section, this kind of algorithm has a large number of parameters that need to be correctly tuned in order to obtain good results. Generally, only experienced users can do this correctly and, moreover, the task of selecting the best choice of values for all these parameters has been suggested to constitute itself an optimization problem [93]. In addition, GAs show a poor performance in some problems (deceptive and separable problems) in which the existing operators of crossover and mutation do not guarantee that better individuals will be obtained changing or combining existing ones.

Some authors [107] have pointed out that making use of the relations between genes can be useful to drive a more "intelligent" search through the solution space. This concept, together with the limitations of GAs, contributed to spread a new type of algorithms grouped under the name of Estimation of Distribution Algorithms (EDAs).

EDAs were introduced in the field of EAs in [156], although similar approaches can be previously found in [261]. In EDAs there are neither crossover nor mutation operators. Instead, the new population of individuals is sampled from a probability distribution, which is estimated from a database that contains the selected individuals from the previous generation. Thus, the interrelations between the different variables that represent

$D_0 \leftarrow$ Generate M individuals and evaluate each of them
Repeat for $l = 0, 1, 2, ...,$ until stopping criterion is met
  $D_l^{Sel} \leftarrow$ Select N individuals from the D_l population following a selection method
  Induce from D_l^{Se} an n (size of the individual) dimensional probability model
  $D_{l+1} \leftarrow$ Generate a new population of M individuals based on the sampling of the probability distribution $p_l(\boldsymbol{x})$ learnt in the previous step
  Evaluate individuals in D_{l+1}

Figure 2.2: Pseudo-code for EDAs.

the individuals may be explicitly expressed through the joint probability distribution associated with the individuals selected at each generation. Figure 2.2 presents a common outline for all EDAs.

A review of different EDAs is presented in the following sections, classified on the basis of the different probability models that can be used to represent the dependencies between the variables that constitute the individuals. Algorithms have been grouped according to the way dependencies between variables are considered: all variables are independent, pairwise dependencies, or multiple dependencies. For the sake of convenience to the thesis objectives, we restrict the review to EDAs for optimization in discrete domains; for an excellent review including continuous variables the reader is referred to [130].

Firstly, some notation that will be used throughout the dicussion is introduced. Given an n-dimensional random variable $\boldsymbol{X} = (X_1, X_2, ..., X_n)$ and a possible instantiation $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, the joint probability distribution of $\boldsymbol{X}$ will be denoted by $p(\boldsymbol{x}) = p(\boldsymbol{X} = \boldsymbol{x})$. In the case of two unidimensional random variables X_i, X_j and their respective possible values x_i, x_j, the conditional probability of X_i given $X_j = x_j$ will be represented as $p(x_i|x_j) = p(X_i = x_i|X_j = x_j)$. In the context of EAs, an individual with n genes can be considered an instantiation $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. Let the population of the l-th generation be D_l. The individuals selected, D_l^{Se}, constitute a dataset of N cases of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. EDAs estimate $p(\boldsymbol{x})$ from D_l^{Se}, therefore, the joint probability distribution of the l-th generation will be represented by $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se})$.

2.3.1 Without Dependencies

All the models that belong to this category consider all variables as independent. Therefore, the joint probability distribution is factorized as a product of univariate and independent probability distributions. That is, $p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i)$.

UMDA

Univariate Marginal Distribution Algorithm (UMDA). Introduced in [149], this algorithm uses the simplest way to estimate the joint probability distribution:

$$p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_l(x_i) \tag{2.3}$$

where each univariate marginal distribution is estimated from marginal frequencies:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i|D_{l-1}^{Se})}{N} \tag{2.4}$$

being

$$\delta_j(X_i = x_i|D_{l-1}^{Se}) = \begin{cases} 1 & \text{if in the } j^{th} \text{ case of } D_{l-1}^{Se},\ X_i = x_i \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

UMDA has been successfully applied to different problems: feature subset selection [4], learning of Bayesian networks from data [23; 200], or to solve linear and combinatorial problems using Laplace correction [176].

Other works focus on the behavior of the algorithm, performing a mathematical analysis of UMDA [152; 153], studying its convergence when UMDA is used to maximize a number of pseudo-boolean functions [90], or analyzing the genetic drift phenomenon [109; 110; 225].

Finally, several modifications have also been introduced in UMDA trying to improve its performance: modifications on the simulation phase [213; 214], use of a repair method for solving constraint satisfaction problems [95], adaptive population sizing [108], use of memory schemes for dynamic optimization problems [256], or introducing the bitwise mutation operator [96].

BSC

Bit-Based Simulated Crossover (BSC) [236] originated as a recombination operator for GAs. This approach uses the fitness value of the selected individuals to estimate each marginal distribution:

$$p_l(x_i) = \frac{\sum_{\{\boldsymbol{x}|\delta_j(X_i=x_i|D_{l-1}^{Se})=1\}} e^{f(\boldsymbol{x})}}{\sum_{\{\boldsymbol{x}\epsilon D_{l-1}^{Se}\}} e^{f(\boldsymbol{x})}} \tag{2.6}$$

where function δ_j maintains the meaning expressed in Equation 2.4.

In Equation 2.6, the numerator alludes to the sum of the evaluation function values of the individuals with value x_i in the variable X_i, and the denominator is the sum of evaluation values of the selected individuals.

This algorithm has been applied to problems such as feature subset selection [113] and partition clustering tasks [202].

PBIL

Population Based Incremental Learning (PBIL) [11; 12] uses a probability vector to represent the characteristics of the population:

$$p_l(\boldsymbol{x}) = (p_l(x_1), \ldots, p_l(x_i), \ldots, p_l(x_n)) \tag{2.7}$$

where $p_l(x_i)$ refers to the probability of obtaining a value of 1 in the i^{th} variable of the l^{th} population.

The vector is initialized using the first population, and then it is used to sample a new set of M individuals. From this set, only the best N individuals are selected. We denote them by:

$$\boldsymbol{x}^l_{1:M}, \ldots, \boldsymbol{x}^l_{i:M}, \ldots, \boldsymbol{x}^l_{N:M} \tag{2.8}$$

Based on the following Hebbian inspired rule, the probability vector is updated:

$$p_{l+1}(\boldsymbol{x}) = (1-\alpha)p_l(\boldsymbol{x}) + \alpha\frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}^l_{k:M} \tag{2.9}$$

where $\alpha \in (0, 1]$ is a parameter of the algorithm (the reader may note that when $\alpha = 1$, this algorithm performs as UMDA).

The following population will be sampled from this new (updated) probability vector. In contrast to the general EDA behavior, it must be remarked that this algorithm uses the probability vector of the previous generation in addition to the recently sampled individuals to obtain the new probability vector.

PBIL has been applied to different problems, such as: optimization of parameters of a solution in the field of tactical driving [234], search for optimal weights in a neural network structure [47; 81], classifier selection [205], optimization of parameters for the simple supply chain model [91], or learning of Bayesian networks [23].

Some theoretical studies of PBIL have been completed in [89; 106].

Finally, there are works that use characteristics of PBIL or even modify parts of the algorithm. In [135], the Statistical and Inductive Tree Based Evolution algorithm is presented. This approach mixes ideas from PBIL (probability vector) with inductive

For $i = 1, \ldots, n$
  If $x^l_{i,1:2} \neq x^l_{i,2:2}$
    If $x^l_{i,1:2} = 1$
      $p_l(x_i) = p_{l-1}(x_i) + \frac{1}{K}$
    If $x^l_{i,1:2} = 0$
      $p_l(x_i) = p_{l-1}(x_i) - \frac{1}{K}$

Figure 2.3: Process to update the probability vector in cGA. K is a constant value fixed as a parameter.

decision trees. In general terms, it works as follows: starting with a randomly created population, individuals are split into three groups (best, mediocre, and bad) and Induction of Decision Trees is used to induct a decision tree, extracting the set of equivalent rules. Then, PBIL is used together with the rules to sample new individuals. The process is repeated until some termination criteria is fulfilled.

Related to dynamic problems, there are two different proposals: using a dual probability vector and competing with the main probability vector to generate samples [258], and using a memory scheme to store the best sample and the working probability vector [257].

cGA

The compact Genetic Algorithm (cGA) [98] is quite similar to PBIL. It also uses a probability vector to guide the search through the space of possible solutions.

This algorithm completes the following steps: first, the probability vector is initialized (each component follows a Bernoulli distribution with parameter 0.5). Then, two individuals are randomly sampled from the probability vector and evaluated. Taking into account their fitness value, one of them will be the best ($x^l_{1:2}$) and the other the worst ($x^l_{2:2}$). The process to update the probability vector is presented in Figure 2.3.

It must be noted that the probability vector is updated in an independent way for each variable. This process of adaptation of the vector of probabilities towards the winning individual continues until the vector of probabilities has converged.

cGA has been applied to feature subset selection [35] and to the pruning of neural networks used in classification problems [36].

A runtime analysis of cGA using different linear functions is presented in [62].

Finally, several modifications on cGA have been presented in the literature. In [80], a modified compact GA is developed for the intrinsic evolution of continuous time recurrent neural networks. In [1], two elitism-based cGAs are presented: persistent elitist compact

genetic algorithm (pe-cGA), and nonpersistent elitist compact genetic algorithm (ne-cGA).

RELEDA

The Reinforcement Learning Estimation of Distribution Algorithm (RELEDA) was introduced in [177].

In this algorithm an agent explores an environment perceiving its current state as well as information about the environment. Based on that information, the agent makes some decisions, making the environment change and receiving the value of this transition as a scalar reinforcement sign.

This algorithm is similar to UMDA, but the probability of each variable is updated applying a reinforcement learning method. The search for probability distributions is reduced to a number of parameters denoted by $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ where $\theta_i \in \Re$ is a parameter related to the probability of the variable X_i through a function. The correlation between $p(x_i)$ and θ_i is expressed through the sigmoid function:

$$p(x_i) = \frac{1}{2}(1 + \tanh(\beta\theta_i)) \tag{2.10}$$

where β is the sigmoid gain.

In each generation, the value of the parameters θ_i is modified by a Δ_i value following:

$$\Delta\theta_i = \alpha(b_i - p(x_i))(1 - d_i) \tag{2.11}$$

$$b_i^{t+1} = \gamma b_i^t + (1 - \gamma)x_i \tag{2.12}$$

where b_i is the reinforcement signal (baseline), d_i is the marginal distribution of the variable X_i, x_i is the value of the variable X_i in the best individual in that generation, α is the learning rate, and γ is the baseline factor.

This algorithm has been compared in [177] to other EDAs (UMDA and PBIL) using two well-known problems: four peaks and bipolar function, showing that it requires fewer fitness evaluations to obtain an optimal solution.

DEUM

Distribution Estimation Using MRF with direct sampling (DEUM) [223]. This algorithm uses the Markov Random Field (MRF) modelling approach to update the probability vector. It can be seen as an adaptation of the PBIL approach by replacing marginal frequencies with an MRF model on a selected set of solutions.

For $i = 1, \ldots, n$
  If $\alpha_i < 0$
    $p_i = p_i(1 - \lambda) + \lambda$
  If $\alpha_i > 0$
    $p_i = p_i(1 - \lambda)$

Figure 2.4: Process to update the probability vector in DEUM. λ is a learning rate (values between 0 and 1) fixed as a parameter.

In [29], MRF theory was used to provide a formulation of the joint probability distribution that relates solution fitness to an energy function calculated from the values of the solution variables. Mathematically:

$$p(x) = \frac{f(x)}{\sum_y f(y)} = \frac{e^{-U(x)}}{\sum_y e^{-U(y)}} \tag{2.13}$$

therefore

$$-\ln(f(x)) = U(x) \tag{2.14}$$

where $f(x)$ is the fitness function of an individual and $U(x)$ an energy function that specifies the joint probability distribution. Generally, the energy function involves interaction between variables but, for this particular approach, all the variables are considered independent. Therefore, the previous equation can be rewritten as:

$$-\ln(f(x)) = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \tag{2.15}$$

Each solution in any given population gives an equation satisfying the model. Therefore, selecting N promising solutions from a population allows us to estimate the distribution by solving $A\alpha = \boldsymbol{F}$, where A is the $N \times n$ dimensional matrix of values in the selected set, $\boldsymbol{\alpha}$ is the vector of MRF parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, and $\boldsymbol{F}$ is the N dimensional vector containing the value $-\ln(f(x))$ of the selected set of solutions.

Finally, the probability vector will be updated using the MRF parameters (see Figure 2.4).

This algorithm has been modified in [224], proposing an approach called DEUM$_d$ in which a MRF model is directly sampled to generate the new population.

2.3.2 Pairwise Dependencies

Algorithms in this second group only consider dependencies between pairs of variables. In this way, estimation of the joint probability can still be done quickly. However, it must

be noted that an additional step is required (not necessary in the previous algorithms): the construction of a structure that best represents the probabilistic model.

MIMIC

Mutual Information Maximization for Input Clustering (MIMIC) [53]. This approach searches (in each generation) for the best permutation between the variables. The goal is to find the probability distribution, $p_l^\pi(\boldsymbol{x})$, that is closest to the empirical distribution of the set of selected points, $p_l(\boldsymbol{x})$, when using the Kullback-Leibler divergence, where

$$p_l^\pi(\boldsymbol{x}) = p_l(x_{i_1} \mid x_{i_2}) \cdot p_l(x_{i_2} \mid x_{i_3}) \cdots p_l(x_{i_{n-1}} \mid x_{i_n}) \cdot p_l(x_{i_n}) \tag{2.16}$$

and $\pi = (i_1, i_2, \ldots, i_n)$ denotes a permutation of the set of indexes $\{1, 2, \ldots, n\}$.

The Kullback-Leibler divergence between two probability distributions, $p_l(\boldsymbol{x})$ and $p_l^\pi(\boldsymbol{x})$, can be expressed as:

$$H_l^\pi(\boldsymbol{x}) = h_l(X_{i_n}) + \sum_{j=1}^{n-1} h_l(X_{i_j} \mid X_{i_{j+1}}) \tag{2.17}$$

where

$$h(X) = -\sum_x p(X = x) \log p(X = x) \tag{2.18}$$

denotes the Shannon entropy of the X variable, and

$$h(X \mid Y) = \sum_y h(X \mid Y = y) p(Y = y) \tag{2.19}$$

where

$$h(X \mid Y = y) = -\sum_x p(X = x \mid Y = \ y) \log p(X = x | Y = y) \tag{2.20}$$

denotes the mean uncertainty in X given Y.

Therefore, the problem of searching for the best $p_l^\pi(\boldsymbol{x})$ can be solved by searching for the permutation π^* that minimizes $H_l^\pi(\boldsymbol{x})$.

As a search over the $n!$ possible permutations will be unfeasible for most of the problems, a greedy search is proposed to find the π^* permutation. The process starts with the variable X_{i_n} with the smallest estimated entropy. In the following steps, the variable with the smallest average conditional entropy with respect to the variable selected in the previous step is chosen (obviously from the set of variables not yet chosen).

MIMIC has been used to solve several problems: the traveling salesman problem [199], feature subset selection [113], partial abductive inference problem in Bayesian networks [54], or learning of Bayesian networks [200].

In addition, some modifications of this algorithm have also been proposed, applying a repair method for solving constraint satisfaction problems [95], or introducing a mutation operator [96].

COMIT

Combining Optimizers with Mutual Information Trees (COMIT) [13]. This algorithm hybridizes the EDA approach with local optimizers. Estimation of the probability distribution of the selected individuals in each generation is made using a tree structured Bayesian network, learnt using the algorithm Maximum Weight Spanning Tree (MWST) proposed in [41].

In general terms, MWST looks for the probabilistic tree structure ($p_l^t(x)$) that best matches the probability distribution of the selected individuals ($p_l(x)$). To consider the quality of each possible tree, the Kullback-Leibler cross-entropy measure is used. The distance is minimized by projecting $p_l^t(x)$ on any MWST, where the weight of the branch (X_i, X_j) is defined by the mutual information measure:

$$I(X_i, X_j) = \sum_{x_i, x_j} p_{(X_i,X_j)}(x_i, x_j) \log \frac{p_{(X_i,X_j)}(x_i, x_j)}{p_{X_i}(x_i) p_{X_j}(x_j)} \tag{2.21}$$

Once an estimation of $p_l(\boldsymbol{x})$ has been obtained, COMIT samples a number of individuals from it and selects the best as the initial solutions of a local search method. The resulting individuals are then used to create a new population.

TREE

TREE [130] refers to an adaption of COMIT where the local search step is eliminated. Thus, new individuals to enter the next population are created directly by sampling the distribution that estimates $p_l(\boldsymbol{x})$.

TREE has been applied to several problems. To name a few, the traveling salesman problem [199], feature subset selection [112], partitional clustering [202], rule induction [72; 227] and software test data generation [210].

BMDA

Bivariate Marginal Distribution Algorithm (BMDA) [187]. This algorithm uses a factorization of the joint probability distribution that only needs second-order statistics.

It is based on an acyclic (but not necessarily connected) dependency graph. This graph is constructed as follows: first, a variable is chosen arbitrarily and it is added as a node of the graph. This first variable is the one with the greatest dependency on the rest of the variables –measured by Pearson's χ^2 statistic.

Secondly, the variable with the greatest dependency between any of those previously added and the set of those not yet added is incorporated to the graph. This second step is repeated until there is no dependency surpassing a previously fixed threshold between already added variables and the rest. If this is the case, a variable is chosen at random from the set of those not yet used to create a new tree structure. The whole process is repeated until all variables are added into the dependency graph.

In each generation the factorization obtained with the BMDA is given by:

$$p_l(\boldsymbol{x}) = \prod_{X_r \in R_l} p_l(x_r) \prod_{X_i \in V \setminus R_l} p_l(x_i \mid x_{j(i)}) \tag{2.22}$$

where V denotes the set of n variables, R_l denotes the set containing the root variable –in generation l– for each of the connected components of the dependency graph, and $X_{j(i)}$ returns the variable connected to the variable X_i and added before X_i.

2.3.3 Multiple Dependencies

Different works [26; 183] have shown the limitations of using simple approaches to solve difficult problems. It must be noted that in these kinds of problems, different dependency relations can appear between variables and, hence, considering all of them independent or taking into account only dependencies between pairs of variables may provide a model that does not represent the problem accurately.

Several algorithms have been proposed in the literature using statistics of order greater than two to factorize the probability distribution. In this way, dependencies between variables can be expressed properly without any kind of initial restriction. However, it must be also noticed that the probability model required for some problems could be excessively complex and, sometimes, unaffordable in computational terms.

ECGA

Extended Compact Genetic Algorithm (ECGA) [97]. This algorithm divides the variables into a number of groups (clusters) which are considered independent. Therefore, in each

generation, the factorization of the joint probability distribution is expressed as a product of marginal distributions of variable size. These distributions are related to the variables that are contained in the same group and to the probability distributions associated with them. In this way, the factorization of the joint probability distribution on the n variables is:

$$p_l(\boldsymbol{x}) = \prod_{c \in C_l} p_l(\boldsymbol{x}_c) \tag{2.23}$$

where C_l denotes the set of groups in the l^{th} generation, and $p_l(\boldsymbol{x}_c)$ represents the marginal distribution of the variables $\boldsymbol{X}_c$, that is, the variables that belong to the c^{th} group in the l^{th} generation.

The grouping is carried out using a greedy forward algorithm that obtains a partition between the n variables (as mentioned above, each group of variables is assumed to be independent of the rest).

The process starts considering n clusters (one variable in each cluster) and then continues trying to unify the pair of clusters that reduce the most a measure value. This value conjugates the sum of the entropies of the marginal distributions with a penalty for the complexity of the model based on the minimum description length principle (MDL) [198].

ECGA has been applied to feature subset selection [35] and to the pruning of neural networks used in classification problems [36].

From a theoretical point of view, in [217] empirical relations for population sizes and convergence times are presented.

Finally, some modifications of this algorithm have been proposed. In [133], a hybrid ECGA that combines crossover and mutation operators. The proposed algorithm combines the Building Blocks-wise crossover operator from ECGA with a recently proposed Building Blocks-wise mutation operator that is also based on the probabilistic model of ECGA [218]. In [216], a sub-structural niching method is proposed and applied to ECGA aiming to maintain diversity at the sub-structural level.

FDA

Factorized Distribution Algorithm (FDA) [155]. It must be noted that this algorithm differs from the others in regard to the probabilistic model. Instead of creating a new one at each generation, the same model is used throughout the entire execution. Therefore, this algorithm needs the factorization and decomposition of the task to be given by an expert – which is not a common situation. Generally, due to this characteristic, it is intended to be applied to additively decomposable functions for which, using the running intersection property [131], a factorization of the mass-probability based on residuals, x_{b_i}, and separators, x_{c_i}, is obtained.

$BN_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ where S_0 is an arc-less DAG, and $\boldsymbol{\theta}^0$ is uniform
$p_0(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$
$D_0 \leftarrow$ Sample M individuals from $p_0(\boldsymbol{x})$
For $l = 1, 2, \ldots$ until the stopping criterion is met
  $D_{l-1}^{Se} \leftarrow$ Select N individuals from D_{l-1}
  $S_l^* \leftarrow$ Find the best structure according to a criterion:
    • penalized maximum likelihood+search (if EBNA_{BIC})
    • penalized Bayesian score + search (if $\text{EBNA}_{\text{K2+pen}}$)
    • conditional (in)dependence tests (if EBNA_{PC})
  $\boldsymbol{\theta}^l \leftarrow$ Calculate θ_{ijk}^l using D_{l-1}^{Se} as the data set
  $BN_l \leftarrow (S_l^*, \boldsymbol{\theta}^l)$
  $D_l \leftarrow$ Sample M individuals from BN_l using PLS

Figure 2.5: Pseudo-code for the EBNA_{BIC}, $\text{EBNA}_{\text{K2+pen}}$ and EBNA_{PC} algorithms.

The joint probability distribution can be factorized as:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{k} p_l(\boldsymbol{x}_{b_i} | \boldsymbol{x}_{c_i}) \tag{2.24}$$

As this factorization remains valid for all the iterations, the only changes are those in the estimation of probabilities.

Theoretical results for FDA can be found in [150; 151; 152; 153; 155; 260]. In addition, the space complexity of the algorithm is studied by [82] using random additive functions as the prototype.

EBNA

In this section, three different algorithms (EBNA_{PC}, $\text{EBNA}_{\text{K2+pen}}$, and EBNA_{BIC}), grouped under the name of Estimation of Bayesian Networks Algorithms (EBNAs), are presented. Introduced in [68; 129], their main characteristic is that the factorization of the joint probability distribution is encoded by a Bayesian network, learnt from the database containing the selected individuals in each generation. A common scheme for these approaches can be seen in Figure 2.5.

Before explaining the different variations of EBNAs, we proceed with a brief introduction to Bayesian networks that will be helpful to better understand the algorithms.

Bayesian networks Formally, a Bayesian network [40] over a domain $\boldsymbol{X} = (X_1, \ldots, X_n)$ is a pair (S,$\boldsymbol{\theta}$) that represents a graphical factorization of a probability distribution. The

structure S is a Directed Acyclic Graph (DAG) which reflects the set of conditional (in)dependencies between the variables. The factorization of the probability distribution is codified by S:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i|\boldsymbol{pa_i}) \tag{2.25}$$

where $\boldsymbol{Pa_i}$ is the set of parents of X_i (variables from which there exists an arc to X_i in the graph S). In Figure 2.6 for example, $\boldsymbol{Pa_3}$=$\{X_1,X_2\}$ (X_1 and X_2 are the parents of X_3).

The second part of the pair, $\boldsymbol{\theta}$, is a set of parameters for the local probability distributions associated with each variable. If variable X_i can take r_i possible values, $x_i^1, \ldots, x_i^{r_i}$, the local distribution, $p(x_i|\boldsymbol{pa_i^j},\boldsymbol{\theta}_i)$ is an unrestricted discrete distribution:

$$p(x_i^k|\boldsymbol{pa_i^j},\boldsymbol{\theta}_i) \equiv \theta_{ijk} \tag{2.26}$$

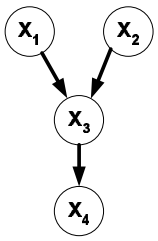
where $\boldsymbol{pa_i^1},\ldots,\boldsymbol{pa_i^{q_i}}$ denote the values of $\boldsymbol{Pa_i}$ and the term q_i denotes the number of possible different instances of the parent variables of X_i. In other words, parameter θ_{ijk} represents the conditional probability of variable X_i being in its k^{th} value, knowing that the set of its parent variables is in its j^{th} value. Therefore, the local parameters are given by $\boldsymbol{\theta}_i = (((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})\ \ i = 1,\ldots,n$. An example of a Bayesian network can be seen in Figure 2.6.

In the context of EDAs, EBNAs comprise a group of algorithms that use Bayesian networks to codify the dependencies between variables. At each generation, given a set of individuals (population), a Bayesian network must be learnt trying to properly reflect the relations between variables. After that, the Bayesian network is sampled in order to obtain the new population.

Related to the learning process, there are mainly two different methods: "score + search" and "detecting conditional (in)dependencies".

"score + search": This method uses a score (metric) to measure the quality of the Bayesian network. Among the different scores used, we can point out the Bayesian Information Criterion (BIC) [221] or the Bayesian Dirichlet equivalence (BDe) [102]. Once the Bayesian network has been assigned a score, the goal is to complete a search step, changing the structure of the Bayesian network with the aim of improving the current score.

Generally, the search step begins with an empty Bayesian network (without arcs) and, in the following steps, arcs will be added based on the score used to measure the quality of the network. In order to have an effective algorithm, it is necessary to find an adequate model as soon as possible (even if it is not optimal).



Structure

Local probabilities

$$\begin{aligned}
\boldsymbol{\theta_1} &= (\theta_{1-1}, \theta_{1-2}) && p(x_1^1), p(x_1^2) \\
\boldsymbol{\theta_2} &= (\theta_{2-1}, \theta_{2-2}, \theta_{2-3}) && p(x_2^1), p(x_2^2), p(x_2^3) \\
\boldsymbol{\theta_3} &= (\theta_{311}, \theta_{321}, \theta_{331}, && p(x_3^1|x_1^1, x_2^1), p(x_3^1|x_1^1, x_2^2), p(x_3^1|x_1^1, x_2^3), \\
&\quad \theta_{341}, \theta_{351}, \theta_{361}, && p(x_3^1|x_1^2, x_2^1), p(x_3^1|x_1^2, x_2^2), p(x_3^1|x_1^2, x_2^3), \\
&\quad \theta_{312}, \theta_{322}, \theta_{332}, && p(x_3^2|x_1^1, x_2^1), p(x_3^2|x_1^1, x_2^2), p(x_3^2|x_1^1, x_2^3), \\
&\quad \theta_{342}, \theta_{352}, \theta_{362}) && p(x_3^2|x_1^2, x_2^1), p(x_3^2|x_1^2, x_2^2), p(x_3^2|x_1^2, x_2^3) \\
\boldsymbol{\theta_4} &= (\theta_{411}, \theta_{421}, \theta_{412}, \theta_{422}) && p(x_4^1|x_3^1), p(x_4^1|x_3^2), p(x_4^2|x_3^1), p(x_4^2|x_3^2)
\end{aligned}$$

Factorization of the joint mass-probability

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)$$

Figure 2.6: Structure, local probabilities and resulting factorization for a Bayesian network with four variables (X_1, X_3 and X_4 with two possible values, and X_2 with three possible values).

For example, Algorithm B [31] is a common method used to learn Bayesian networks. This algorithm uses a hill climbing strategy. Starting with an arc-less structure, it adds in each step the arc that maximizes the score. When no improvement can be achieved, the algorithm stops. An alternative to Algorithm B could be the use of the model created in the previous generation, instead of beginning each time with an empty structure.

Some of the algorithms that belong to this group are EBNA_{BIC} and $\text{EBNA}_{\text{K2+pen}}$. Both use Algorithm B as a search method, but EBNA_{BIC} uses the BIC score to measure the quality of the Bayesian network, and $\text{EBNA}_{\text{K2+pen}}$ combines the Bayesian approach to calculate the marginal likelihood [44] with a penalizing term, introduced to avoid an excessively complex Bayesian network.

"detecting conditional (in)dependencies": The techniques that belong to this group complete several tests to detect the relations between variables. These algorithms usually start with the complete undirected graph, and then independence tests are performed to remove edges. When no more edges can be removed, an orientation process is completed to create the Bayesian network. For example, EBNA_{PC}, one of the algorithms that belongs to this family, uses the *PC* algorithm [232] to detect the dependencies. Starting with the complete graph, it is "thinned down" by re-

moving edges with zero order conditional independence relations, "thinned" again using first order conditional relations, then second order conditional relations are taken into account, and so on. The set of variables conditioned on need only to be a subset of the set of variables adjacent to one of the variables of the pair. The independence test is performed based on the χ^2 distribution. When there are no more tests to do, the orientation process begins, giving a direction to each edge in the graph.

Finally, once the Bayesian network has been learnt, new individuals are sampled to create the new population. Among the different methods, EBNAs use the Probabilistic Logic Sampling method [103]. In this method, the instances are generated one variable at a time in a forward way. That is, a variable is sampled after all its parents have already been sampled. To do that an ancestral ordering of the variables is given $(\pi(1), \ldots, \pi(n))$ where parent variables are before children variables. Once the values of $\boldsymbol{Pa}_{\pi(i)}$ have been assigned, we simulate a value for $X_{\pi(i)}$, using the distribution $p(x_{\pi(i)}|\boldsymbol{pa}_{\boldsymbol{\pi(i)}})$.

EBNA approaches have been applied to several problems; for instance, graph matching [19], partial abductive inference in Bayesian networks [54], feature subset selection [111; 112], job scheduling problem [138], rule induction [227], traveling salesman problem [199], partitional clustering [202], knapsack problems [206] or software testing [210; 211].

In [95] EBNA was modified by applying a repair method for solving constraint satisfaction problems, and in [96] a mutation operator is introduced.

Parallel approaches for EBNA_{BIC} and EBNA_{PC} have been presented in [139; 144]; a parallel and multi-objective version of EBNA_{BIC} to solve a chemical problem is shown in [145].

In [82], the space complexity of the EBNA algorithm has been studied using random additive functions.

BOA

Bayesian Optimization Algorithm (BOA) [179; 180; 183; 184; 185] uses a "score + search" method (B Algorithm) to construct the model, using as a metric the Bayesian Dirichlet equivalence (BDe) [102]. In each generation, the process starts with an empty structure. In order to reduce the cardinality of the search space, the number of parents that each node can have is limited to k.

This algorithm has been extended and applied to several problems. In [181] BOA is modified in order to model hierarchical problems using a type of hybrid model called a Huffman network. In [185] it is adapted to include local structures by using decision graphs to guide the network construction.

Other extension named Mixed BOA that uses decision trees with mixed decision nodes is presented in [164]. In [161] MBOA is combined with variance adaptation in order to improve its behavior in the continuous domain.

Some theoretical studies have been completed using Bayesian networks to estimate the fitness of the individuals [188] or to reduce the number of parameters needed to execute the BOA algorithm [186].

The real-coded Bayesian Optimization Algorithm (rBOA) algorithm is proposed in [2], as an extension of BOA to the area of real-value optimization. It performs a factorization of a mixture of probability distributions, and finds maximal connected graphs (substructures) of the factorization graph (probability model). Then, it fits each substructure independently by a mixture distribution estimated for clustering results in the corresponding partial-string space. Finally, offspring is obtained by a sampling method based on independent subspaces.

Two parallel approaches have been presented for BOA using a pipelined parallel architecture [162] and clusters of computers [163]. Recently, in [165] the parallelization of the learning of decision trees using multi-threaded techniques has been proposed.

The different BOA approaches have been applied to feature subset selection [35], to the pruning of neural networks used in classification problems [36], ising spin-glass systems and maximum satisfiability problems [182]. In [117; 118] a comparative review of some EAs (including MBOA) is presented, evaluating them on a different number of test functions in the continuous domain.

LFDA, FDA$_L$, FDA-BC, FDA-SC

Learning Factorized Distribution Algorithm (LFDA), introduced in [151], essentially follows the same steps as EBNA$_{\mathrm{BIC}}$. The main difference is that in the LFDA the complexity of the model learnt is controlled by the BIC metric in conjunction with a restriction on the maximum number of parents that each variable can have in the Bayesian network.

An initial algorithm FDA$_L$ is proposed in [168], to learn – by means of conditional (in)dependence tests – a junction tree from a database. The underlying idea is to return the junction tree that best satisfies the previous independences, once a list of dependencies and independencies between the variables is obtained.

Also, in [166], a structure learning algorithm that takes into account questions of reliability and computational cost is presented. The algorithm, called FDA-BC, studies the class of Factorized Distribution Algorithm with Bayesian networks of Bounded Complexity.

Similar ideas are introduced in the FDA-SC [167]. In this case the factorization of the joint probability distribution is done using simple structures, i.e. trees, forests or polytrees.

PADA

Polytree Approximation of Distribution Algorithms (PADA) [229]. The factorization is done using a Bayesian network with polytree structure (no more than one undirected path connecting every pair of variables). The proposed algorithm can be considered a hybrid between a method for "detecting conditional (in)dependencies" and a procedure based on "score + search".

MN-EDA

Markov Network Estimation of Distribution Algorithm (MN-EDA) [212]. The authors introduce a method that approximates probability distributions using what they call "messy factorizations". In order to learn the factorizations, the algorithm combines a reformulation of a probability approximation procedure used in statistical physics (Kikuchi approximations), with a novel approach for selecting the initial inputs required by the procedure.

In addition, a new method for sampling solutions from the approximation is also used (Gibbs Sampling). The learning and sampling methods are the primary components of this MN-EDA.

2.3.4 Other Algorithms

- An EDA in the permutation representation domain that uses Edge Histogram Based Sampling Algorithms (EHBSAs) is presented in [241]. The algorithm starts generating random permutation strings for each individual in the population. Then, individuals are evaluated and the most promising solutions are used to construct a symmetrical Edge Histogram Matrix (EHM) where an edge is a link between two variables in an individual. Finally, new individuals will be sampled from that EHM, replacing the old population. The behavior of the algorithm is tested on the traveling salesman problem.

- Estimation of Distribution Programming (EDP) is presented in [254]. This program is codified using a probabilistic graphical model given by a Bayesian network. The search method follows the common scheme of EDAs to solve Genetic Programming applicable problems. This work is extended in [255], where the proposed EDP is mixed with a GP algorithm.

- Dependency Detection for Distribution Derived from *df* (DDDDD or D^5) [240]. This approach combines EDAs with linkage identifications in order to detect dependencies. It has three parts: (1) calculation of fitness differences – each variable is perturbed and then fitness difference for the perturbation is calculated –, (2)

classification of individuals according to the fitness difference, and (3) estimation of the classified individuals based on entropy measures.

- The algorithm presented in [253] uses marginal frequencies to constrain the estimated probability distribution. A schema is a subset of the search space where the values of some variables are defined (fixed) and the values of the others are variable (represented by *). The order of the schema is defined by the number of *. Given a frequency distribution over the search space and a schema, the corresponding schema frequency is just the sum of the relative frequencies of the elements of that schema.

  The entropy of this distribution is subsequently maximized and the distribution is sampled to produce a new population. In this work, only contiguous order-2 schema families are used, proposing as a future work the use of higher order schemas.

- In [194] a Learning Automata based Estimation of Distribution Algorithm (LAEDA) is presented. This algorithm follows the general EDA scheme, and uses a variable structure learning automata as the probability model.

- Finally, Unsupervised Estimation of Bayesian Network Algorithm (UEBNA) is introduced in [189]. This approach uses a Bayesian network for data clustering in order to factorize the joint probability distribution of the individuals selected at each iteration. The goal of this approach is to optimize multimodal problems.

2.4 Scatter Search

The Scatter Search (SS) technique [85; 123; 125] is presented in the literature as a novel instance of EAs. Though this method shares with EAs some of their features, it also sets a number of fundamental differences. In fact, principles of SS were established by concepts developed independently from the evolutionary paradigm.

According to Glover [85], the notion of combining solutions or rules to create new solutions originated in the 1960s. Researchers in the field of scheduling proposed the merging of rules to obtain improved local decisions. Such an approach was motivated by the conjecture that information about the relative desirability of a choice is captured in different ways by alternative rules. This notion was soon extended to the field of mathematical relaxation for optimization, where the creation of *surrogate constraints* was devised through a heuristic [84] which was, in turn, the stem of SS.

In the same manner as EAs maintain a population of individuals, SS operates on a set of solution points, the *reference set*, by combining them to create new solutions. Hence, both methodologies assume that solutions encode useful information about the problem, and that this information is transferred to new solutions when merged. On the other

hand, the main conceptual dissimilarity lies in the management of the diversification and intensification notions. While in EAs, selection, recombination and mutation of individuals are probabilistic, in SS, selection and combination of solutions follow systematic strategies. Moreover, intensification may be forced through the application of a heuristic improvement procedure to each new solution, and the diversity in the reference set can be explicitly controled during the search.

The following discussion attempts to introduce basic concepts of SS. Detailed descriptions, together with more sophisticated extensions, can be found in the book by Laguna and Martí [125] or in any of the excellent reviews available in the bibliography, e.g. [85].

2.4.1 Basic SS Scheme

The SS algorithm departs from the construction of a set P of solutions to guarantee a critical level of diversity. In other words, this phase promotes the generation of solutions increasing the diversity in P. Optionally, a heuristic method is applied to each solution before entering the set; if so, a local search is generally employed. Next phase of the algorithm consists of an iterative process. In the first round, the reference set, $RefSet$, is built by extracting the best solutions from P. The meaning of "best" in this context is not limited to a measure given exclusively by the objective function. In particular, a solution may be added to $RefSet$ if the diversity of the set is enhanced, though the objective value of such solution is worse than other competitors. In the next step, a number of subsets of solutions is systematically generated. The members of these subsets are combined to generate new solutions that might replace others in $RefSet$. As in the initial phase, new solutions are, optionally, improved with a local search method before considering their inclusion in $RefSet$. The "best" solutions (broad brush meaning, once again) are added to $RefSet$. If a new solution has been included, new subsets are generated and the process repeats. Otherwise, the algorithm finishes.

Structurally, a SS algorithm is composed of the following five interacting methods. The functionality of each method is clearly specified. However, its definition remains open to the problem being solved, which grants this technique a suitable flexibility.

Diversification Generation Method A method that generates a number of diverse solutions.

Improvement Method Once a solution is obtained, this method aims at improving it, usually through a local search method. Although this method is not strictly required, the common trend is to include it in the SS methodology.

Reference Set Update Method This method manages $RefSet$ by defining the strategies necessary to build and update it. Both, building and updating, may be based

$P \leftarrow \emptyset$
$P \leftarrow$ Add $|P|$ distinct solutions obtained by diversification and improvement
$RefSet \leftarrow$ Add the b_1 solutions in P with best objective function value and delete them from P
$RefSet \leftarrow$ Add the b_2 most diverse solutions in P in relation to the solutions in $RefSet$
Repeat while new solutions are in $RefSet$
  Generate all new subsets of solutions from $RefSet$
  Obtain new solutions by combination and improvement
  $RefSet \leftarrow$ Update $RefSet$ with new solutions

Figure 2.7: Pseudocode of basic SS.

on the objective function value, the diversity between solutions or an alternative criterion. If no new solution is added to $RefSet$, the algorithm stops. Nonetheless, in many cases, a maximum number of iterations is established in order to avoid too long executions.

Subset Generation Method The subsets of solutions are systematically generated from $RefSet$. At least, all subsets formed by two solutions are created. As the number of subsets tends to be high, there is a need for keeping $RefSet$ small; generally, $|RefSet| = |P|/10$.

Solution Combination Method This method creates new solutions by combining the solutions in a given subset.

The interaction of the five methods can be observed in the basic SS algorithm proposed in Figure 2.7. A common size for P is 100 solutions and, therefore, $|RefSet| = 10$. Notice the improvement method has been included in the algorithm, though it is an optional component. A classical strategy for constructing the reference set is to select from P the $b_1 = |RefSet|/2$ solutions with the best objective function value, and the remaining b_2 most diverse solutions. As noticeable, the subset generation method only considers the new subsets associated with the solutions introduced in the previous step. If the maximum number of iterations is not reached and no solution has been added to $RefSet$, then the process halts.

Figure 2.8 presents a schematic illustrating the roles of the SS methods, assuming improvement is applied. Circles represent new solutions, uncoloured before the application of the improvement method, and black afterwards.

2.4.2 Advanced SS Designs

The advanced features of SS are related to the way the five methods described above are implemeted. In other words, the sophistication level is given by the implementation of the SS methods, instead of the decision to include or exclude some elements from the

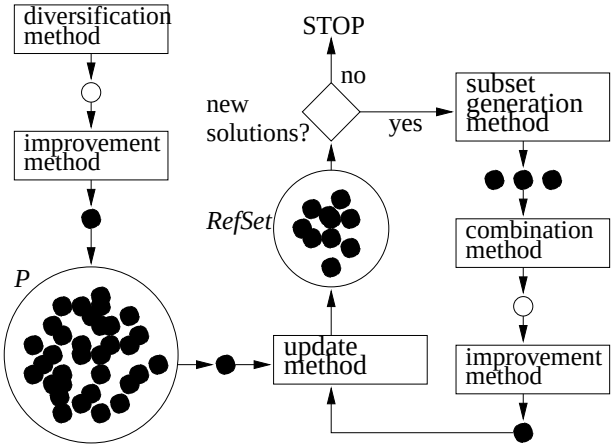


Figure 2.8: Schematic of a basic SS design [125].

approach. Next, a few interesting advanced strategies are described in brief; details may be consulted in [125].

Reference set rebuilding

The basic SS process finishes when no new solution is added to $RefSet$. This implies the algorithm has converged, since no new solution would be generated from a further combination. A possibility for scaping from such a situation could lie in the injection of diversity in $RefSet$. Thus, if no solution is added to the set, a common practice is to perform a rebuilding step and run the algorithm once again. For instance, a simple rebuiding strategy consists of creating a new set P and replacing half of the worst solutions in $RefSet$ with the solutions in P which most increase the diversity in $RefSet$. As a result of such strategy, the SS algorithm is extended as shown in Figure 2.9.

Reference set dynamic update

In the basic design from Figure 2.7, new solutions that are to become members of $RefSet$ are not combined until the next iteration of the algorithm. This strategy is known as *static update*. On the other hand, the *dynamic update* strategy applies the combination method to new solutions in a manner that is faster than in the basic design. That is, if a new solution is to be admitted in $RefSet$, the goal is to allow this new solution to be subjected to combination as quickly as possible. For this, the solution is inmediately included in the $RefSet$, instead of waiting for the rest of *parent* solutions to be combined.

$P \leftarrow \emptyset$
$P \leftarrow$ Add $|P|$ distinct solutions obtained by diversification and improvement
$RefSet \leftarrow$ Add the b_1 solutions in P with best objective function value and delete them from P
Repeat for $l = 1, 2, ..., MaximumIteration$
  $RefSet \leftarrow$ Add the b_2 most diverse solutions in P in relation to the solutions in $RefSet$
  $NewSolutions \leftarrow TRUE$
  Repeat while $NewSolutions = TRUE$
    $NewSolutions \leftarrow FALSE$
    Generate all new subsets of solutions from $RefSet$
    Obtain new solutions by combination and improvement
    $RefSet \leftarrow$ Update $RefSet$ with new solutions
    If $RefSet$ changed
      $NewSolutions \leftarrow TRUE$
    Else
      $RefSet \leftarrow$ Delete the b_2 solutions with worst objective function value from $RefSet$
      $P \leftarrow \emptyset$
      $P \leftarrow$ Add $|P|$ distinct solutions obtained by diversification and improvement

Figure 2.9: Pseudocode of extended SS.

Multiple solutions combination

The combination mechanism in SS is not limited in its general form in combining just two solutions. However, the mechanism cannot handle all subsets of size i, $i \in \{1, 2, ..., |RefSet|\}$ (there are $2^N - N - 1$ subsets, with $N = |RefSet|$). A procedure to control the total number of subsets consists of a strategy to expand pairs into subsets of larger size. The following approach selects representative subsets of different sizes by creating subset types:

- Subset Type 1: all 2-element subsets.
- Subset Type 2: 3-element subsets derived from the 2-element subsets by augmenting each subset of type 1 to include the best solution not in this subset.
- Subset Type 3: 4-element subsets derived from the 3-element subsets by augmenting each subset of type 2 to include the best solution not in this subset.
- Subset Type 4: the subsets consisting of the best i solutions, $i \in \{5, 6, ..., |RefSet|\}$.

2.4.3 Performance Results

Although based on mathematical foundations and classical methods, SS suffers, likewise other metaheuristics, from a lack of theoretical works. Nonetheless, this technique is nowadays one of the centres of attention of the optimization community. Its suitability

is mainly due to the increasing number of successful aplications in a wide variety of problems. Improved benchmarks for solving such problems have resulted from these applications, along with new advances for solving a significant range of real-life situations.

Just to name a few examples, in [126] ϵ-optimal solutions were obtained for 30 from up to 40 multimodal function optimization problems. Moreover, the SS design showed to find solutions in fewer evaluations than a GA. In [141], several implementations of SS are compared with GAs to solve four black-box permutation problems, resulting in a slight superiority of the formers. The linear ordering problem was dealt with in [33]. A number of diversification procedures are studied and show to be competitive when compared to other classical methods. SS has also been applied to practical optimization problems like neural network training, arc crossing minimization in graphs, maximum clique problem, graph coloring, vehicle routing or job-shop scheduling; see [125] for discussions on these works. More recently, the knapsack problem [48] and software test data generation [211] have been faced using SS.

The procedures employed in the previous works, as well as in others, have yielded a number of SS designs differing from the basic template. Taking the results obtained by these designs into account, lessons for future developments are presented in [124].

Regarding the diversification generation method, it is suggested in this work that the use of a memory structure to create solutions provides a proper balance between diversity and quality. By contrast, while a pure random method generates highly diverse solutions, their quality tends to be low.

A conclusion related to the improvement method concerns its influence on the computational complexity of the SS algorithm. This method may imply such an overload that the investigation of its selective use is proposed. Additionally, the application of improvement to every solution accelerates the convergence of $RefSet$, suggesting this method should be studied from a methodological perspective as well.

A hint which may be useful for further approaches is that solution quality is more important than diversity when updating $RefSet$. Albeit the possible strategies for this step, according to the experimental results, it seems that the best performance is not achieved if diversity is used as a main updating criterion.

Apropos the subset generation method, it has been observed that most of the searching power can be atributed to the combination of 2-solution subsets. In [33], different subset types were empirically employed, one after the other. The outcomes showed that at least 80% of the solutions to enter $RefSet$ came from combinations of 2-element subsets. Nonetheless, this result should be carefully taken, as a distinct sequence of subsets combination could modify this percentage.

Finally, a lesson stated in [124], about the combination method, is that the use of multiple strategies can be effective. This is inspired by GAs implementations where good results

have been attained by generating new individuals from recombination and mutation operators.

2.5 Other Recent Metaheuristics

The field of optimization has been experiencing a resurgence of procedures in the last years, mainly from the area of metaheuristics. Though these are out of the scope of the present thesis, we find it interesting to draw some comments on the subject, since they are modern optimization techniques that are deserving the attention of researches in some contexts.

The Greedy Randomized Adaptive Search Procedure (GRASP) [70] combines in an iterative process a solution construction with a local search. At the construction step, a feasible solution is iteratively built in a semi-greedy way. A set of possible element candidates to be part of the solution is recalculated at each generation. These candidates are a "piece" of the induced solution. One element is selected and added to the solution. The element candidate list is evaluated with respect to a greedy scoring function in order to select the next element to be added to the construction. The evaluation of the elements is used to create a list, which consists of the best. The element to be added into the partial solution is randomly chosen from the list. Once an element is included in the partial solution, the list is updated. The solution induced is then applied a local search method. A particularly appealing characteristic of GRASP is that it is easy to implement and, usually, a small number of parameters is needed [125].

Ant Colony Optimization (ACO) [61] is a search method that mimics the foraging behavior of ants. Ants deposit an amount of pheromone on the ground, thus influencing the choices of other members. The larger the load of pheromone in a path, the higher the probability that an ant selects this path. In ACO, pheromone is seen as a heuristic value that is assigned to partial solutions based on the frequency of its presence in good solutions. As the construction of the new solutions is carried out by using an auxiliary probabilistic value based on the pheromone value, there is a bias in the algorithm to form solutions which contain building blocks that have shown to be good in previous steps.

Roughly speaking, the idea behind the Variable Neigborhood Search (VNS) [147] is a local search where the neighborhood is systematically changed. VNS explores increasingly distant neighborhoods of the current solution at each step of the process. More precisely, a solution is drawn at random from the current neighborhood of the current solution and a local search is applied departing from this neighbor. If the resulting solution improves the current one, then the current best is updated and the process restarts; otherwise, a wider neighborhood is tried.

3 Fundamental Concepts on Software Testing

Testing is a crucial part of the software development process. It plays a main role in the search for the quality required as it constitutes the primary way used in practice to verify the correct behavior of the software produced. One of the most important issues in software testing is the generation of the input cases to be applied to the program under test. Due to the expensive cost of this task, its automatization has become a key aspect. A number of options for this has been proposed under the name Search Based Software Test Data Generation. The aim of such approaches is the creation of test data by means of heuristic search optimization methods. More precisely, most developments over the last years have concentrated on metaheuristic methods, offering promising results.

This chapter is devoted to the introduction of such approaches. Firstly, the motivation for software testing is presented. Different aspects of the test process, together with the classical alternatives for test data generation, are briefly overviewed next. Finally, the field of input generation methods based on heuristic search is dealt. Owing to the extensive scope of the field, the discussion is intended to provide insights on the basic elements to achieve the automatic obtention of test data.

3.1 Software Quality

Considering the crucial role software plays nowadays, *quality* assurance becomes a main issue in the field. Software is so deeply present in daily life that the effects of an undesirable behavior can be dramatic [231], even for human beings [132]. In contrast to other products, such as manufacturing goods, where a balance between productivity and quality is sought, in software development, these two concepts are almost indistinguishable [17]. Moreover, complexity of software systems is continuously growing in order to exploit the huge advances in computer hardware, resulting in an increasing development cost. As some authors state [114; 115], no other product in the industrialized world is as labor-intensive and error-prone as software. In fact, software quality has been suggested to be the most critical and difficult technological challenge of modern times [20; 58; 59].

The elusive concept of software quality may be defined either from a technical or customer oriented perspective. From the technical side, quality is the fulfillment of the specified

requirements [190; 231]. From the customer point of view, quality is the conformance of software to the user needs or expectations [67; 231][1]. Regardless of the perspective, definition remains extremely vague, as the meaning of terms "requirement", "expectation" or "need" connotes a subjective evaluation. We require more specific means of assessing whether software quality has been achieved or not. Thus, requirements or expectations are represented by a number of desirable software characteristics, and quality attainment consists then of their satisfaction. Usually, description of a characteristic is still not precise enough so that it can be quantified. There are however related attributes which can be measured to express the degree of excellence in this characteristic, allowing to elicite the achieved quality level [92].

In order to settle a standard basis, researchers and organizations, such as ISO and IEEE, have developed models that describe quality characteristics and their interrelations [63; 171; 74]. Despite the lack of consistency and unity in some of the terms [190], as well as in the characteristics involved and their treatment, a few elements are common to most of the approaches. Hence, characteristics such as *usability* (extent to which the software is praticable to use), *maintainability* (capability of updating) or *reliability* are usual among quality models [63]. Reliability is defined as the probability that software functions without *failure* for a given period of time under specified conditions [191]. Description of a software failure is an area for open debate; we resort to IEEE [170]. An *error* refers to a mental mistake made by the programmer or designer. The manisfestation of that error in the code is called a *fault*. The occurrence of an incorrect output resulting from an input value that is received with respect to the specification is named a failure. Quality is mainly influenced by failures [190], so reliability is considered to be the most important software characteristic [59; 63; 67]. Indeed, it is a prerequisite of other properties, e.g. usability, and it is often mistakenly used as a synonym of quality [67].

Owing to the fact that a population of identical software systems, operating under similar environmental conditions, fail at different points in time, failure phenomena are typically explained in probabilistic terms. Furthermore, as, in general, the whole set of faults in a program is unknown, true reliability cannot be elicited, so it is estimated, mainly through probabilistic models. Certain models try to assess the number of faults in a program, while others study the *failure rate* (failures per unit time in a time interval) or the number of observed failures by time t. Some approaches measure and predict the improvement of reliability during the software development process or even take environmental factors, such as the programmer skill, into account. Anyhow, most of the models in the literature require a considerable amount of failure data to estimate their parameters [25]. For an interesting formal description of reliability models the reader is referred to [191].

Basically, attempts to improve software reliability consist of preventing or reducing faults introduced during the development process. A common way of fault prevention is to fo-

[1] It is worth to emphasize both definitions since one does not necessarily imply the other.

cus on the most complex modules in a system and to assign them larger resources. On the other hand, fault reduction involves software verification, detection and correction of faults. So, improvement efforts can be applied at the different stages of the so-called *software lifecycle* [24], that is, *analysis* (requirements and functional specifications), *design*, *coding*, *testing* and *operating*. Among these, testing is the most significant with regard to reliability [17; 59; 67; 191].

3.2 Software Testing

Testing may be described as the process of executing a software program to expose failures [17; 67]. In other words, testing consists of operating the program with an *input* and checking whether the obtained behavior is correct or not. An input refers to an assignment of values to the program *parameters*, which are, in turn, the set of variables whose values need to be fixed to enable an execution.

The high relevance of testing with respect to reliability comes as a consequence that both concern with failures. As the primary way of failure detection [67], testing becomes crucial for reducing faults in the software. In addition, it represents a powerful fault prevention method, since the knowledge on the system and the reasonings carried out to create a test may avoid errors [17]. Indeed, testing is not only significant for improving reliability, but also for its evaluation. The measurement of software reliability cannot be performed without previously discovered failure data [191]. Moreover, testing intrinsically involves a validation process, so it serves as a means of gaining confidence that the software is reliable enough.

Nonetheless, the rest of phases from the software lifecycle are still needed to improve reliability, i.e. testing by itself is not sufficient [59]. Finding an input revealing a failure may be extremely difficult. The software conditions that trigger a fault can be remarkably complex, a fault might remain latent for a period of time only to arise in a particular environment, or it could even be masked by other faults [78]. Thus, due to the intrincate dynamic behavior of faults, not all the failures in a program are usually detected. Anyhow, in order to ensure that all failures have been found, the whole set of program inputs, i.e. the *input domain*, should be checked. Although finite in essence, such input domain is often so huge that a complete exploration results unfeasible. In fact, this inability for an exhaustive validation constitutes the most basic limitation of testing: it can show the presence, but not the abscence of faults [60]. This implies testing is not able to provide a proof that the software is correct. The alternative approach for demonstrating the correctness of software is *formal verification*, that is, mathematical proofs that the system meets all the conditions required of it. However, this method also suffers from some well-known disadvantages which fairly restrict its application [17]. Just to name

two, it shows a lack of scalability to the complexity of modern programs, and each of the mathematical proofs is in turn subject to failures.

Therefore, in spite of its drawbacks, testing remains the foremost mechanism in practice for detecting failures and verifying the correct behavior of software. In consequence, it is a major way of improving and assessing reliability and, hence, attaining quality [59]. Quantitative arguments from the real world also support this relevance. Testing usually accounts for 50% of the project resources [17; 24], growing up to 80% in some safety-critical systems [28]. Even so, a recent study [237] estimated the U.S.A. users suffer annual economic losses derived from software faults totaling more than $59.5 billion. More important from the standpoint of software developers, however, is the finding that more than a third of those losses could have been saved via better testing.

3.3 Test Process

The test process involves a large number of activities, strategies and elements which make testing a vast field. Next, we point out a few ideas that are useful to introduce the following discussions. An immense literature on software testing topics exists; for instance, the reader might consult the classical book by Beizer [17] or the more recent one by Kaner et al. [116].

Given a software program, testing is generally applied at different levels, each built on the last:

Unit testing A unit refers to the minimal software module that can be tested, e.g. in object-oriented programming, a class method. Thus, at this level, a unit is tested in isolation from the rest of the system.

Component testing A component is a module formed of a number of units, e.g. in object-oriented programming, a class. Obviously, a unit is a component, and so is the whole system.

Integration testing At this level, the purpose of a test is to expose failures in the interfaces and dependencies between software components.

System testing These tests are oriented towards the verification that the complete software system meets its requirements.

Acceptance testing It alludes to the user validation. This involves testing the software system under simulated real-world operating conditions as well as delivering the so-called *beta* program versions to a limited audience.

Typically, testing proceeds from unit to acceptance level, since the cost of correcting a fault grows superlinear in this direction [67]. A further reason is that up to 65% of the failures may be detected with unit testing [17].

Being one of the stages from the software lifecycle, testing is an arranged process including many activities. For example, system requirements need to be studied, tests must be designed and executed, results observed, and conclusions reported. Each activity is in turn subject to faults, so, if a failure is found, both the software and the test process should be explored for the cause. Moreover, during the correction of a fault, new ones could be injected in the software or others which were previously masked could arise. This implies once the fault is fixed, tests need to be processed again, yielding an iterative procedure; such a situation is known as *regression testing*.

Among the previous activities, the generation of a set of *test cases* is of great importance. A test case can be described as a piece of information concerning one software execution. This information includes an input with which the program will be executed, the expected program behavior for that input, and any additional useful data for processing the test, e.g. an identification number. Then, it may be inferred that a mandatory task within this activity is the generation of a set of inputs to be applied to the program under test.

3.4 Generation of Test Inputs

As remarked above, exhaustive testing is generally prohibitive due to the huge size of the input domain. Furthermore, real-world demands and modern development tools are dramatically increasing the productivity of programmers, so claiming an increasing amount of testing in less and less time. Thus, tests are designed with the purpose of addressing particular aspects of the software system. This makes the generation of a set of test inputs a non-trivial task, as it must be adequate to the test type and its requirements.

Input generation methods mainly conform to two testing strategies [75]: *random* and *subdomain based*. In this context, random testing alludes to the employment of inputs sampled from the input domain according to a probability distribution. For instance, due to its simplicity, a well-known approach is to generate test data simulating a uniform distribution on the input domain [64]. A more sophisticated alternative, instead, is to employ the *operational profile* of the program, that is, the expected run time probability distribution of the inputs [157]. The purpose of this approach, then, is to test the program in a way close to its real usage. However, input generation is not straightforward, as knowledge on the system is needed to estimate the probability distribution, which can be a costly task [94]. On the other hand, the underlying idea in subdomain based testing is specifying subsets from the input domain, called *subdomains*, and requiring the set of inputs to include an element from each of the subsets. A common assumption for the

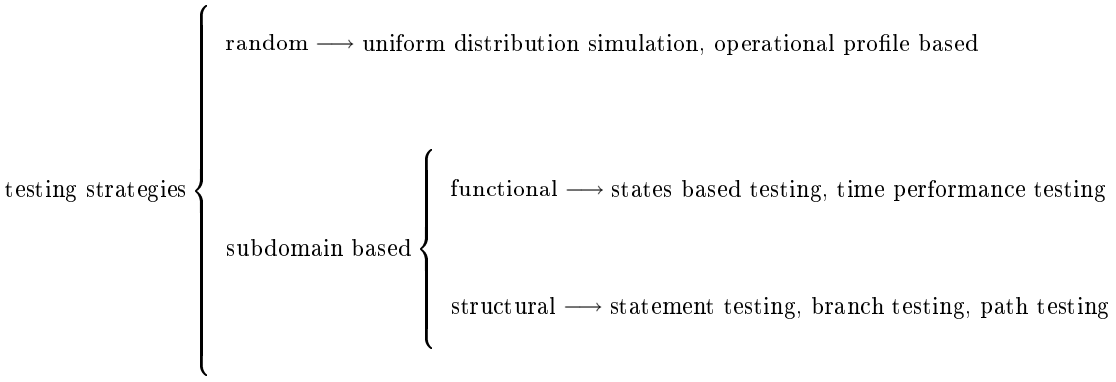


Figure 3.1: Scheme of usual testing strategies.

subdomains is that their union must lead to the input domain, so in the particular case where they are disjoint, the strategy is known as *partition testing* [249]. In any case, it is important to note that, rather than explicitly, subdomain specification is often implicitly driven by the purpose of the particular testing approach.

Considering the criteria used for splitting the input domain, subdomain based testing may be further classified as *functional* or *structural* [18; 75]. In functional (also known as *black box*) testing, each subdomain consists of the inputs satisfying a condition or combination of conditions asserted in the program specification. Therefore, the aim of this strategy is to test aspects regarding the functionality of software. For example, a general approach is assigning a subdomain to each of the functional states of the program [173]; test data generation consists then of finding a set of inputs that visits each state. A more specific functional criterion is *time performance testing* [18], which tests whether the response time constraints of the program are fulfilled or not, i.e. there is a subdomain for each time constraint. By constrast, structural (or *white box*) testing relies on the intuition that faults are exposed if certain parts of the source code execute. More precisely, in structural approaches, subdomains come specified by the so-called *code coverage criteria*. For example, in *statement testing* [17], the inputs implying the execution of a code statement (instruction) describe a subdomain. Consequently, a set of inputs must be generated so that each statement is covered. Other coverage criteria will be discussed in detail below. Figure 3.1 shows a basic scheme of the testing strategies just described, together with some of their instances as examples.

3.4.1 Code Coverage Criteria

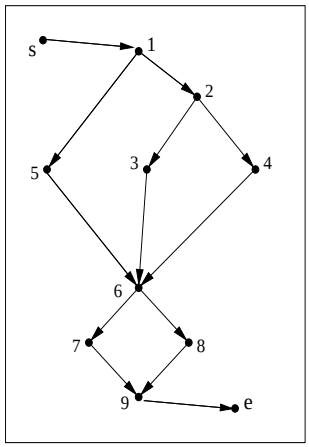
Structural testing is probably the most widely used class of strategies to test programs [17; 159]. Based on the assumption that a fault is exposed when certain portions of code are executed, code coverage criteria are defined in order to detect as many failures as possible.

The source code of the program reveals different control or data flow entities, such as statements, *branches*, *paths*, *defs*, *p-uses* or *c-uses*. The first three examples are control flow entities, while the others are data flow ones. A branch refers to one of the possibilities for the flow of control from a conditional statement in the code. A path, instead, is a sequence of statements that the control flow may traverse. A def alludes to a *definition*, that is, the assignment of a value to a program variable. A variable is *used* if its value is fetched; an use in a conditional statement is called a p-use, and if it is elsewhere in the code, it is named a c-use.

Structural entities of a program can be represented by means of a graph. Although many alternatives exist in the literature [101], we will restrict our attention to the *control flow graph* [69]. A control flow graph $G = (X, U)$ is defined by a set X of vertices and a set $U \subseteq X \times X$ of arcs. Each vertex in X denotes a *code basic block*, excepting two vertices labeled s and e, which refer to the program entry and exit. A code basic block is a maximal sequence of code statements such that if one is executed, then all of them are. An arc $(x, y) \in U$, with x and y distinct from s and e, is such that the control of the program can be transferred from block x to y without crossing any other block. Analogously, for every arc $(s, x) \in U$ or $(y, e) \in U$, it will be possible to transfer the flow of control from the entry to block x and from block y to the exit, respectively. We call a vertices sequence $x_1, x_2, ..., x_n$, with $(x_{i-1}, x_i) \in U, \quad \forall i \in \{2, ..., n\}, \ n \in \mathbb{N}$, a path from x_1 to x_n.

In this kind of graph, a statement is then represented by a vertex, a branch by an arc (x, y) where $outdegree(x) > 1$, and a program path by a path from s to e in the graph. Instead, to reflect defs, c-uses or p-uses associated to a program variable, the graph needs to be enlarged with appropriate labels on the vertices. Such a modified control flow graph is sometimes referred to as a *data flow graph* [49]. For a vertex x labeled u, if $outdegree(x) > 1$, then a p-use is represented, otherwise a c-use associates. So, given a program variable, a definition or a use in a code basic block might be reflected by respective d or u labels in the corresponding vertex. Figure 3.2 illustrates a source code together with derived control and data flow graphs. The source code corresponds to a function, written in the *C* programming language [119], which, given three integers representing the coefficients of a quadratic equation, elicites an integer-valued solution, if it exists. The control flow graph of the function is shown in the middle of the figure, and the data flow graph regarding variable `x` in the code, on the right side.

```
      int quad (int a, int b, int c)
1     {
       double d=0, x=0;
       if (a!=0)
2       if ((b*b)−(4*a*c)<0)
3        x=0;
4       else {
         d=(b*b)−(4*a*c);
         x=(−b+(int)sqrt(d))/(2*a); }
5      else
        x=(−c)/b;
6      if ((a*x*x+b*x+c)==0)
7       printf("integer solution: %d",x);
8      else
        printf("no integer solution");
9     }
```



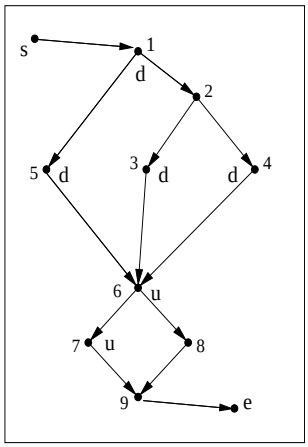


Figure 3.2: Example source code, its associated control flow graph (middle) and enlarged control flow graph (right).

A code coverage criterion specifies a group of structural entities which have to be exercised with a set of program inputs. Several coverage criteria have been developed in the literature so far, which leads to different structural testing approaches. A few well-known instances are described next; a more exhaustive list can be consulted in [17; 76; 77; 116].

- In *statement coverage*, every code statement is chosen to be exercised by a set of program inputs, i.e. the whole set of vertices in the corresponding control flow graph must be covered.
- *Branch coverage* is a classical criterion stating that every branch in the source code must be exercised at least once. Thus, in the associated control flow graph, this implies the coverage of every arc (x, y) with $outdegree(x) > 1$.
- *All c-uses coverage* involves the defs and c-uses of all the program variables. According to this criterion, the set of inputs must exercise, for every variable, a *definition-clear path* from each def to each c-use. Given a def and a use (either c-use or p-use) of a variable `v`, a definition-clear path is a sequence of statements between def and the use such that no other definition of `v` is contained in it. From the graphical point of view, this criterion settles that, in the data flow graph $G = (X, U)$ of each program variable, $\forall\{x_1, x_2\} \in X$ with x_1 labeled d and x_2 labeled u and $outdegree(x_2) \leq 1$, a path $x_1, y_1, ..., y_n, x_2$ where y_i is not labeled d, $\forall i \in \{1, ..., n\},\ n \in \mathbb{N}$, must be found.
- Similarly to the previous criterion, *all p-uses coverage* requires the set of program inputs to exercise, for every variable in the code, a definition-clear path from each

def to each p-use. The analogy in the data flow graph $G = (X, U)$ of each variable, is that $\forall\{x_1, x_2\} \in X$ with x_1 labeled d and x_2 labeled u and $outdegree(x_2) > 1$, a path $x_1, y_1, ..., y_n, x_2$ where y_i is not labeled d, $\forall i \in \{1, ..., n\},\ n \in \mathbb{N}$, must be found.

- *Path coverage* is the most demanding coverage criterion, as all the paths in the program structure are considered for execution. That is, in the associated control flow graph $G = (X, U)$, every path from s to e needs to be covered.

Complexity of code coverage criteria

Depending on the coverage criterion and the program at hand, generating a set of inputs that satisfies a testing approach may result in an extremely hard task.

Several relations have been proposed in the literature to compare coverage criteria; to name a few, *power*, *narrows*, *probbetter* or *properly covers* [76; 247; 248]. Regarding the relative difficulty of satisfying criteria, the *subsumption* relation is one of the most popular [52; 247]. Informally, given a program and two criteria C_1 and C_2, C_1 subsumes C_2 if any set of inputs which fulfills C_1 also fulfills C_2. Subsumption is reflexive, antisymetric and transitive, so it is a partial ordering. According to this, statement coverage is the easiest amongst the previous criteria while path coverage is the most difficult. Figure 3.3 shows this relation for some classical code coverage criteria. An arc from criterion C_1 to criterion C_2 denotes that C_1 subsumes C_2.

Path coverage is exhaustive in the sense that the whole structure of the program is tested. However, it usually becomes unfeasible due to the prohibitive number of paths; this can be noted just by considering the loops in the graph. Instead, statement testing is the less demanding criterion, though its restriction to the coverage of the code basic blocks is deemed insufficient. Nowadays, branch testing is referred as the minimum mandatory coverage criterion [17].

On the other hand, program computations and semantics determine the inputs exercising a given entity, making the test data generation arbitrarily complex. In fact, not all the entities are exercised often. Moreover, the effect of program semantics may result in an entity whose coverage is impossible. Such a case may occur when the program finishes unexpectedly due to a failure, or when the entity is *infeasible*. An entity is called infeasible if there is no input capable of exercising it. For example, the branch represented by the second `if` statement in the code segment of Figure 3.4 is infeasible, since $y > 0$ and $y < 0$ must occur in order for it to be covered. Unfortunately, the problem of discovering whether an entity is infeasible results undecidable [76; 249], so executable entities cannot be known a priori in every case.

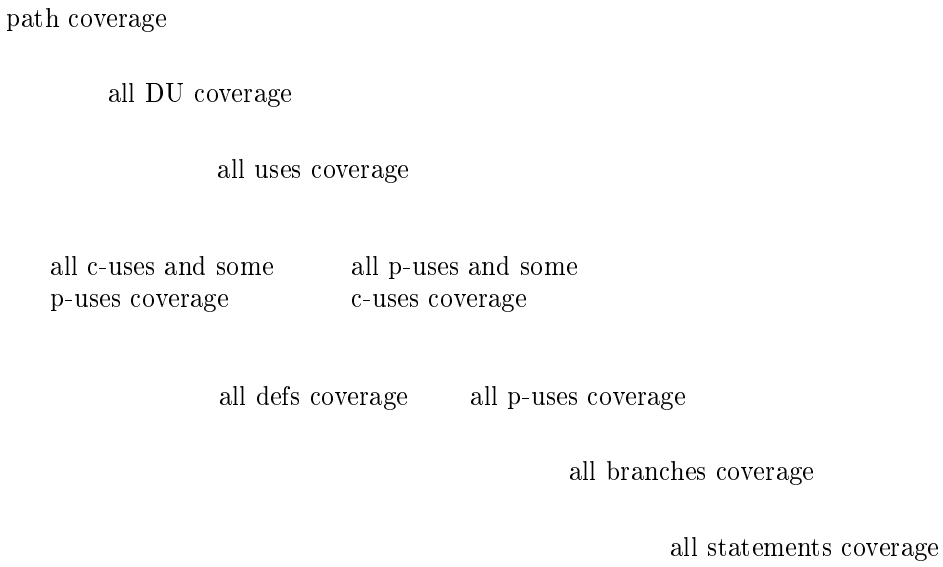


Figure 3.3: Subsumption relation between code coverage criteria.

```
/* previous code segment */
.   .   .   .
if (x==0 && y>0)
        /* basic block where variable y is not defined */
        .   .   .   .   .
        if (y<0)
                /* basic block */
                .   .   .   .   .
/* next code segment */
.   .   .   .   .
```

Figure 3.4: Example of an infeasible branch.

Hence, there is a need to determine the level of completion attained by a set of inputs. This is what the coverage measurement indicates, i.e. the percentage of entities exercised for the particular code coverage criterion.

Automatic test data generation for code coverage criteria

As noted above, the creation of program inputs fulfilling a given code coverage criterion is not trivial. This, together with the fact that in most organizations input generation is performed manually [67; 143], results in a high amount of resources dedicated to such

a task. The automatic generation of test data is hence worthwhile, and some authors suggest it is even vital for the software testing area [174].

Though many are the possibilities, automated structural testing is typically reached by means of random, *static* or *dynamic* input generation methods [71].

A random method relies upon a probability distribution for sampling all the inputs. In spite of its simplicity, the performance obtained tends to be poor for complex programs [71], since the distribution is often chosen without regard to any information on the program at hand. Therefore, the most popular random method so far consists of the uniform distribution, which is used to serve as a basic benchmark for comparison with more sophisticated techniques.

On the other hand, static and dynamic methods are based on knowledge derived from the program structure. The main feature of static methods is that program execution is not required to create test inputs, since they are obtained through a static analysis of the source code. Most of the approaches are inspired by the technique named symbolic execution [42]. This technique consists of choosing an entity from the program structure, and assigning a system of inequalities in terms of the input parameters. The system is built by substituting variables affecting the entity with symbolic values while respecting the constraints associated with the conditions in the code. A solution to the system is then an input exercising the selected entity. In [56], a more recent work using this technique can be consulted. Symbolic execution suffers, however, from well-known problems which limit its performance. The method requires a lot of computational resources, as expressions in the source code have to be resolved and transformed. In case a program variable depends on a function call, no related inequality can be constructed if the source code of the function is unavailable. Furthermore, the resulting system of inequalities could hardly be solved, e.g. if it is nonlinear. Other difficulties arise with array structures, pointers and loops [121; 142].

In contrast to static, dynamic methods execute the program in order to generate the test inputs [121; 146]. While such methods must incur the overheads associated with actually executing the program under test, many of the drawbacks of static methods are overcomed. Moreover, the information available at run-time is exploited to guide the generation of inputs. More precisely, the underlying idea is addressing the automatic generation of test data as an optimization problem [146]. An instrumented version of the program is constructed, i.e. the program is expanded with instructions that will extract information concerning the execution of an input. The collected information is used to assess the closeness of the executed inputs to cover the desired structural entities and guide the search towards new inputs to be executed. In [121], the obtained information determined a function value assigned to each input after execution. The objective was to find an input minimizing its function value, which only occured when reaching the target entity.

Finally, attempts have been developed to combine both the static and dynamic methods. For instance, in [172], a technique called Dynamic Domain Reduction is presented which traverses the control flow graph by symbolically executing the code associated to each vertex.

3.5 Search Based Software Test Data Generation

The automatic generation of test inputs has turned into one of the most challenging problems in the software testing area. An alternative which is deserving the interest of researchers in recent years is Search Based Software Test Data Generation (SBSTDG) [143]. This field alludes to the selection of program inputs making use of heuristic search techniques during the process. The manner in which the heuristic technique takes part remains open, so the optimization point of view in dynamic test data generation is generalized to any other testing approach. In fact, the idea of employing such optimization methods has also been applied in the testing of other manufacturing products, e.g. hardware integrated circuits [46].

Most of the works to date have concentrated on functional and structural testing. Appealing approaches have been proposed for the functional strategy, e.g. safety critical software testing [238] or temporal behavior testing [192]. However, these are out of the scope of the present work, so, in the following, only the structural perspective is discussed, emphasizing branch testing where remarked. A well crafted and extensive review of SBSTDG can be consulted in [143].

3.5.1 The General Scheme

Many of the works developed for structural testing are based on a dynamic test data generation strategy. So, these works consist of choosing the entities to be exercised and, then, searching for the inputs covering them via a heuristic search method. Thus, it is common to more or less follow the general scheme in Figure 3.5. This scheme is an iterative two-step process where, firstly, a previously identified structural entity is selected (a branch, for instance) and marked as an objective. In the second step, the objective entity is assigned a function dependent on the program input, and its optimization is sought. This objective function is formulated in such a way that, if an executed input exercises the objective, the value is optimum. Otherwise, the value is proportional to how close the input is to the objective coverage. Consequently, in order to obtain the function value of an input it must be previously executed on an instrumented version of the program which will provide the information necessary.

This way, the test data generation is tackled as the resolution of a number of optimization problems, one for each objective entity. Early approaches relied on the use of classical

Repeat until stopping criterion is met
  $E \leftarrow$ Select objective entity to exercise
  Obtain input optimizing function for E

Figure 3.5: General scheme for test input generation.

numerical optimization [146] and simple local search methods [121]. By contrast, more recent works resort to global metaheuristics motivated by the fact that the search space defined by the inputs is generally large and complex. Previous classical methods perform poorly in such spaces, as they easily fall into local optima or become unfeasible computationally. Therefore, more sophisticated optimization techniques become a suitable alternative.

For the selection step in Figure 3.5, rather than applying a general rule to determine the objective entity, each approach usually implements a particular alternative [30; 142; 175; 246]. In any case, a common practice is to somehow determine the objective entity with the help of the control flow graph of the program at hand.

The next step of the scheme in Figure 3.5 tackles an optimization problem. That is, given the search space Ω formed by the program inputs and a function $f : \Omega \rightarrow \mathbb{R}$, find $\boldsymbol{x}^* \in \Omega$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})\ \forall \boldsymbol{x} \in \Omega$. From now on, we restrict our attention to the case where entities are branches. Once again, the reader is referred to the survey by McMinn [143] for a discussion about other entities; in [245], a recent study on objective functions for path coverage can be found.

Thus, for branch testing, a classical strategy to create the objective function is the following. Given an objective branch b and an expression $\mathcal{A}\,\mathbf{OP}\,\mathcal{B}$ of the conditional statement **COND** associated with b in the code, with **OP** denoting a comparison operator, the value for an input $\boldsymbol{x}$ is determined by:

$$f(\boldsymbol{x}) = \begin{cases} M & \text{if } \mathbf{COND} \text{ not reached} \\ d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}}) + K & \text{if } \mathbf{COND} \text{ reached and } b \text{ not attained} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where M is the largest computable value, $\mathcal{A}_{\boldsymbol{x}}$ and $\mathcal{B}_{\boldsymbol{x}}$ are appropriate representations of the values taken by $\mathcal{A}$ and $\mathcal{B}$ in the execution, d is a distance measurement, and $K > 0$ is a previously defined constant. Typically, if $\mathcal{A}$ and $\mathcal{B}$ are numerical, then $\mathcal{A}_{\boldsymbol{x}}$ and $\mathcal{B}_{\boldsymbol{x}}$ are their values and $d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}}) = |\mathcal{A}_{\boldsymbol{x}} - \mathcal{B}_{\boldsymbol{x}}|$. In the case of more complex data types, a binary representation of the values for $\mathcal{A}$ and $\mathcal{B}$ can be obtained and, for instance, let $d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}})$ be the Hamming distance [233].

If **COND** involves a compound expression, the overall objective function is constructed from the partial functions for each subexpression. Given two subexpressions C_1 and C_2 with their respective functions f_1 and f_2, and an input $\boldsymbol{x}$, the value for the logical

expression $C_1 \vee C_2$ is $min\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\}$, the logical expression $C_1 \wedge C_2$ is calculated as $f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$, and for $\neg C_1$ the value is known by propagating the negation inside C_1. By applying the associative and commutative properties to different logical expressions, the overall value for f is obtained.

3.5.2 Improving the Objective Function

The previous type of objective function suffers from well-known drawbacks, some of which have no clear solution yet. For example, if the comparison operator in the conditional expression is $\neq$, the function only takes three values and becomes plateau shaped. In order to solve this flaw, several possibilities based on code transformations are described in [100] and [15]. In [27], other weaknesses are identified and a number of alternatives are proposed to overcome them.

To a certain extent, these limitations may be alleviated with the objective function presented in [246]. In addition to the distance in the conditional statement **COND** of the objective branch, a *condition distance* is used for the inputs not reaching **COND**. This distance considers the path from s to e in the control flow graph, taken by an input during program execution. Denoting by v_c the vertex in the control flow graph representing **COND**, and by v_n the nearest previous vertex to **COND** in the path followed by the input, the distance value is calculated in terms of the number of branching vertices straying from the path between v_c and v_n. Therefore, the function in equation 3.1 is extended, maintaining the notation, as follows:

$$f(\boldsymbol{x}) = \begin{cases} d_c(v_c, v_n) & \text{if } \textbf{COND} \text{ not reached} \\ \frac{d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}})+K}{L+(d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}})+K)} & \text{if } \textbf{COND} \text{ reached and } b \text{ not attained} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where d_c is the condition distance and $L > 0$ is a previously defined constant. Notice that L is employed to ensure that the function value when **COND** is not reached surpasses the value when **COND** is reached but b is not attained.

In this manner, if an input was unable to reach the condition, instead of assigning it the worst value (M), the proximity to the condition is taken into account and the objective function is smoothed with regard to equation 3.1.

3.5.3 Applied Metaheuristics and Extensions

Apropos the metaheuristic employed to solve the optimization problem, the most prevalent choice has been the GA. This technique was applied for branch coverage by Sthamer [233] and Wegener et al. [246]. The former compared binary and gray coded representations of the program inputs. However, no clear conclusion could be drawn as to which

of them was superior. In the latter work, excellent coverage results where obtained with a parallel GA using a function of the form of equation 3.2 to calculate the fitness of the individuals. In contrast, in the work by Pargas et al. [175], fitness is only the condition distance described above. GAs have also been chosen for other testing criteria like, for instance, path coverage [134] and *condition/decision* coverage [142]. This last coverage criterion has been recently faced through Evolution Strategies [3]. Metaheuristics proposed in other works include Simulated Annealing [239], Tabu Search [57] and EDAs [210]; all tackling branch coverage with the classical objective function. In [211], Scatter Search was selected for the optimization step. Besides, a collaborative scheme between this method and EDAs was developed.

Although the metaheuristic technique deals with one optimization problem at a time, the real goal of the test case generation is to solve a set of problems. Several approaches in the literature have taken this into consideration to improve the process. The alternative suggested by some works is to profit from the good solutions found by not only evaluating an input for the current objective entity, but also with regard to all the others. Each entity is assigned a set containing the best inputs so far which are used to seed the initial phase of the metaheuristic [246; 210]. Similarly, in [142], the set of an entity is composed of the inputs just reaching the condition associated with the entity. Moreover, this type of strategy is employed for different testing criteria. For instance, the work by Bueno and Jino [30] deals with path coverage, and a set of inputs exercising a selected path is sought at each step; thus, the initial population of a GA is seeded with the closest sets of inputs to covering the path from those stored in a base pool. In contrast, in the approach for path coverage described in [104], a multiobjective optimization view is adopted. This system uses a GA where an individual represents an input and the fitness value is obtained from a weighted sum of the proximities to the coverage of each path. An appealing alternative is developed in [207; 209], where strategies are proposed for searching in the most promising regions of the input space with the aim of enhancing the test data generation process.

Indeed, it should be noted that there are other strategies for structural test data generation, aside from the one outlined in Figure 3.5. For example, in [228], a GA is used once again. However, in this case, an individual corresponds to a set of test inputs, and the fitness is the coverage reached by the set after execution. This way, the problem of generating a set of test cases to fulfill an adequacy criterion is faced from a pure Evolutionary Algorithm view, where an individual represents a solution to the whole problem.

3.5.4 An Example of the General Scheme

To sum up, the preprocessing required to automate the generation of test data for branch coverage following the general scheme in Figure 3.5 should be noticed. Figure 3.6 illustrates this by showing an example program written in the C programming language,

and the elements to be induced from it: the control flow graph and the instrumented program version. The reduced box on the right represents the information supplied by a hypothetical execution of the instrumented program.

The graph is used to select the next objective branch whose coverage will be pursued, for example, branch $(2, 3)$. A GA could be used in the optimization phase. Thus, an individual is a representation of the program input, i.e. three integers. If the inputs set strategy described above is applied, the initial population of the GA could be seeded with the set associated to branch $(2, 3)$.

Each input generated during the search is executed on the instrumented program version in order to elicit its fitness function value. The instrumentation results shown in the reduced box of Figure 3.6 correspond to input $(1, 20, 31)$. The first number in each line of the box contains the traversed basic block and, if the previous block had a condition with an expression $\mathcal{A}\,\mathbf{OP}\,\mathcal{B}$, one more number is included which is the value of $|\mathcal{A}-\mathcal{B}|$ in the execution. Using this information, the value of the condition distance (d_c) shown in equation 3.2 can be obtained. However, this is not necessary, as input $(1, 20, 31)$ reaches the condition of branch $(2, 3)$. Hence, according to equation 3.2 and taking $K = 1$ and $L = 1000$, $f(1, 20, 31) = \frac{276+1}{1000+276+1} = 0.2169$. Although the input is already evaluated for the GA, the instrumentation results are used to calculate $f(1, 20, 31)$ with regard to the rest of the branches. This way, if $(1, 20, 31)$ is a high quality input for a different branch, it is stored in the set of the corresponding branch.

Once the search finishes, a new round of the scheme in Figure 3.5 is performed until, for instance, every branch has been selected as an objective.

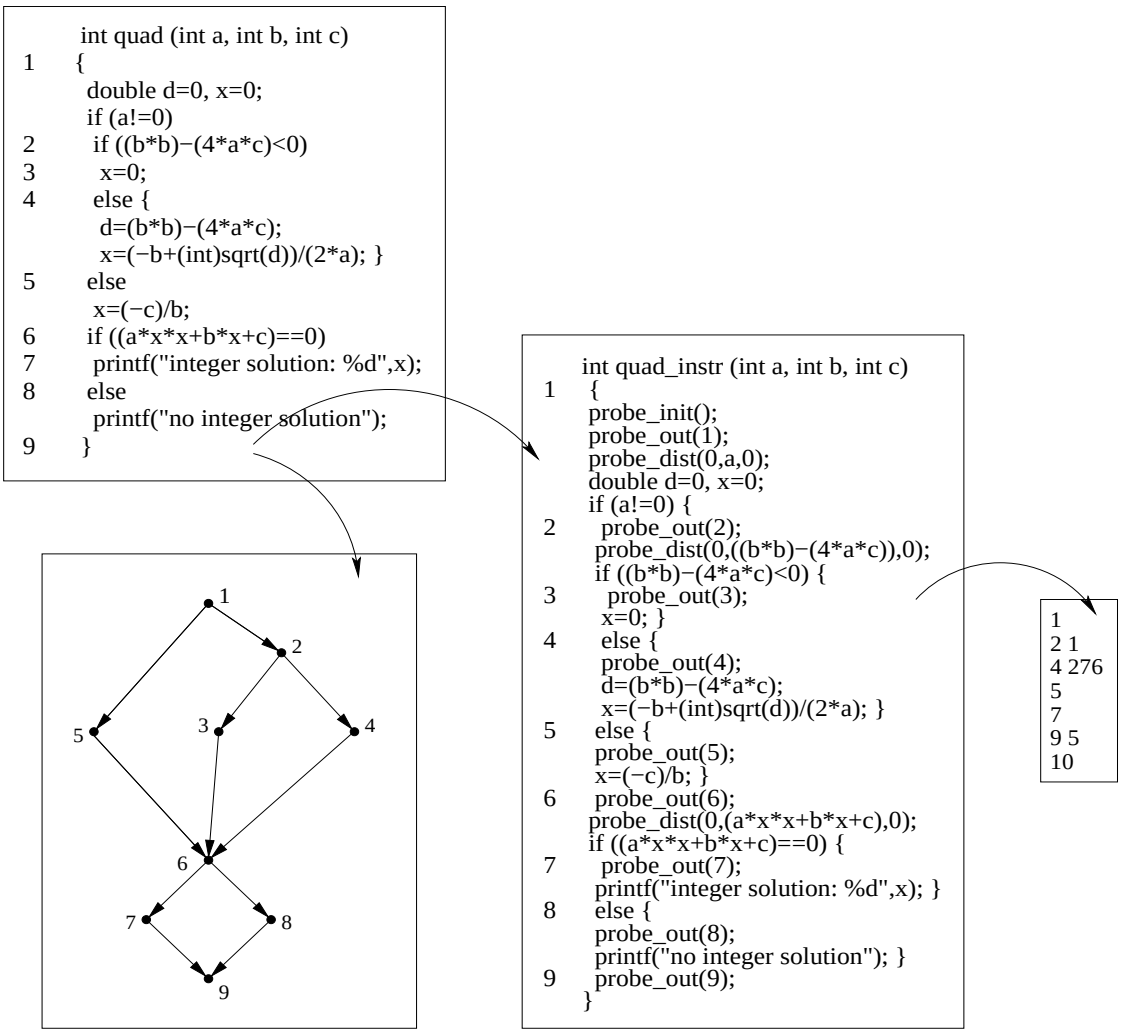


Figure 3.6: Example of source code, control flow graph, instrumented version, and output information.

4 Software Test Data Generation by means of EDAs

One of the most important issues in software testing is the generation of the program inputs used during the test. Particularly, branch coverage is considered a basic criterion to be fulfilled nowadays. On the other hand, EDAs deserve the attention of the Evolutionary Algorithms community, partially supported by the outstanding results obtained in some problems. This chapter is devoted then to the application of EDAs to the problem of finding test inputs for satisfying branch coverage.

Firstly, the system developed for coping with test data generation is explained. Then, the performance of a handful of EDAs, involving several types of probabilistic models, is evaluated through extensive experimentation. In addition, results of EDAs are compared with those of previous works using GAs, yielding interesting conclusions on the adequacy of the former for tackling this problem.

4.1 Motivation

As remarked in the previous chapter, a major issue in software testing is the automatic generation of the inputs to be applied to the program under test. Approaches based on SBSTDG have been offering promising results and, hence, they constitute nowadays a serious alternative to accomplish this task [143]. Until now, works in the SBSTDG literature have concentrated on the use of GAs and, occasionally, on other methods, e.g. Simulated Annealing [120] or Tabu Search [87]. Many other metaheuristics can be exploited however; for instance, most modern techniques could be an appealing alternative.

Thus, considering the high relevance of the test data generation, we deem it worthy to study the application of EDAs. The wide range of possible probabilistic models offered by EDAs turn them into a flexible tool for tackling arbitrarily complex problems. In fact, these metaheuristics have already been applied to several problems with excellent results. Just to name a few works, in [111], Feature Subset Selection was dealt with by means of an EDA which required fewer generations to obtain the same quality results of other Evolutionary Algorithms. In [154], EDAs were compared with other approaches for the resolution of the Graph Bipartitioning problem; EDAs reached the best solution values in all the problem instances.

Indeed, tackling the test data generation is also interesting from the EDAs' point of view as it allows one to evaluate their performance when applied to a demanding and significant real-world problem. By employing alternative probabilistic models, able to reflect different orders of dependencies between variables, it can be checked whether sophisticated EDAs become more adequate than simple ones in this context.

Amongst the different levels at which the test proceeds, unit testing usually accounts for the bulk of the failures detected [17]. Additionally, a common strategy for test data generation consists of obtaining a set of inputs fulfilling a code coverage criterion. Branch coverage is specially relevant, since it is considered the minimum mandatory criterion [17]. So, in the alternative described here, we deal with branch coverage for unit testing of programs writen in the C or C++ language.

4.2 The Optimization Approach

The approach follows a dynamic strategy, that is, the coverage of a branch consists of finding the minimum of a function previously assigned to it. Hence, the test data generation can be posed, in general terms, as a set of optimization problems to be solved.

Each of these problems may be stated as follows: given the input domain Ω and a function $f : \Omega \rightarrow \mathbb{R}$, find $\boldsymbol{x}^* \in \Omega$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})\ \forall \boldsymbol{x} \in \Omega$. The problem is constrained, as Ω is bounded by the finite representation capability of computers and, occasionally, by the program specification. However, due to the arbitrary nature of programs, the rest of characteristics to locate the problem remains open, e.g. input parameters can be scalars or functions, or f can be multimodal or not. Nonetheless, in order to enable the automatization of the process, we will assume a black-box optimization problem, that is, no knowledge is inferred from the objective function. To depart from a simple approximation, f is formulated according to equation 3.1.

4.3 System Framework

The system conforms the general scheme in Figure 3.5. The selection phase follows the option proposed in [246], where a set with the best inputs found so far was associated with a branch during the process, and the branch with the highest quality set was selected as the objective. The optimization step of the scheme allows for the application of several EDAs. Each code branch is associated with one of the three following states: covered, treated but uncovered, and untreated. The stopping criterion is full coverage achievement (all branches in the covered state) or unsuccessful treatment of every unexercised objective branch (branch in the treated but uncovered state).

The system manages infeasible branches like any other one and, therefore, it seeks their coverage. Once the inputs generation process finishes, these branches will be labelled as treated but uncovered, and it might be determined whether their coverage is impossible or whether the system was merely unable to find an input exercising them.

Next, both the optimization and selection steps are described in detail.

4.3.1 Optimization Phase

Given the objective branch, this phase tries to solve the optimization problem raised in Section 4.2 by means of an EDA.

An individual is composed of a 0-1 string representing an input, so that each input parameter is associated a 0-1 substring. In the current implementation of the approach, three parameter types are considered: integers, reals and characters. In the case of an integer, the 0-1 substring represents the parameter following a 2's complement representation. For real numbers, the IEEE floating point codification is used instead, and for a character type, a sign-magnitude codification is employed. In this last case, the number obtained results in a character according to the ASCII code table. The reason for choosing such representation systems [244] relies on the fact that they are usually employed by computers for making the same transformations to internal variables. Since program inputs are to be run in computers, these representations make sense then. Anyhow, for more complex parameter types, an appropriate transformation should be defined to obtain the input parameter value.

The fitness value of an individual is given by function f, defined as in equation 3.1. Information needed to calculate the fitness value is obtained from the performed instrumentation on the program. This instrumentation returns the values of expressions for conditional statements during the execution of the input represented by the individual.

Each branch is bound with a set of individuals which is used as the seed population for the EDA when the branch is the objective. Thus, although the objective branch is fixed, each individual is evaluated according to every other uncovered branch. If the branch is exercised, its state is marked as covered and the input is stored. Otherwise, if the fitness of the current individual is better than the worst individual in the set associated with the branch, then the latter is replaced by the new better individual and, if the branch had previously been treated, its state is marked as untreated. This notion of seeding has also been exploited in other works [142; 246]. Figure 4.1 shows the evaluation algorithm for an individual $\boldsymbol{x_i}$. The value of function f associated with branch b for a given input $\boldsymbol{x}$ is represented by $f_b(\boldsymbol{x})$.

The EDA finishes when either the minimum is found, i.e. an input covering the objective branch is found, or a maximum number of generations is reached. In the first case, the objective branch state is marked as covered and in the second as treated but uncovered.

```
x ← Translate individual x_i to input
Execute instrumented program with x
Repeat for each uncovered branch b
   f_b^i ← f_b(x)
   f_b^w ← Find the fitness of the worst individual x_w in the set associated with b
   If f_b^i < f_b^w
      Substitute x_w by x_i in the set associated with b
      If f_b^i = 0
         Mark b as covered
      else
         Mark b as untreated
   If b is the objective
      fitness ← f_b^i
Return fitness
```

Figure 4.1: Evaluation algorithm pseudocode.

Any EDA approach can be applied. However, assumming the use of the relations between variables benefits the search, multivariate EDAs seem to be more adequate for this problem than simpler alternatives, as the existence of variable interdependencies appears to be evident. Often, input parameters act over program variables which, in turn, interact affecting other variables and so on, leading to non-linear combinations that determine the branches followed by the control flow.

As can be noted, at each generation, a probability distribution is learnt and used to generate new individuals. In other words, the input domain is sampled according to a probability distribution. Therefore, the EDA follows a random test data creation strategy inside each generation. Furthermore, the probability distribution is obtained from the selected individuals, and these are chosen with respect to their fitness, i.e. a dynamic test data generation tactic is adopted. Consequently, this approach can be described as a hybrid between random and dynamic test data generation. This allows for the outlining of the behavior of the referred method, from the testing perspective, as a random generation of inputs which, at each generation, updates its distribution on the basis of the ones already generated.

4.3.2 Selection Phase

A control flow graph is used to identify the branches at the initialization stage and to help decide which branch to select next during the process. Remember that, in a control flow graph, branches come defined by every arc (x, y) with $outdegree(x) > 1$.

```
f̄_best ← ∞
objective ← ∅
tie ← false
Repeat for each untreated branch b
  f̄_b ← Average fitness of the individuals associated with b
  If f̄_b < f̄_best
    f̄_best ← f̄_b
    objective ← b
    tie ← false
  If f̄_b = f̄_best
    tie ← true
If tie = true
  objective ← Breadth first search between branches with f̄_best value
Return objective
```

Figure 4.2: Selection algorithm pseudocode.

Candidate objective branches are those in the untreated state. The branch objective will be the one for which the mean fitness of its associated individuals is the best. In case there is a tie, then a breadth first search is carried out, i.e. from the tied branches, the one with the lowest level in the control flow graph is selected. A pseudocode of the selection algorithm can be observed in Figure 4.2.

The underlying idea is facing the optimization problem with the most promising population seed available at that moment. As one can see, it is possible for a branch already treated to be a candidate objective once again if, during optimization, a new individual is introduced in its set. The reason for this is that the mean fitness in the set is better than in the previous optimization process and could result in a promising population seed.

4.4 An Execution Example

As an illustration of the developed approach, some of the steps of a hypothetical system execution will be explained. The example function used in the previous chapter will be employed once again. Recall that the input of this function consists of three integer-valued parameters. For the sake of clarity, Figure 4.3 presents again the function and the elements to be induced from it.

First of all, the control flow graph is obtained and an instrumented version of the program constructed; both are shown in Figure 4.3.

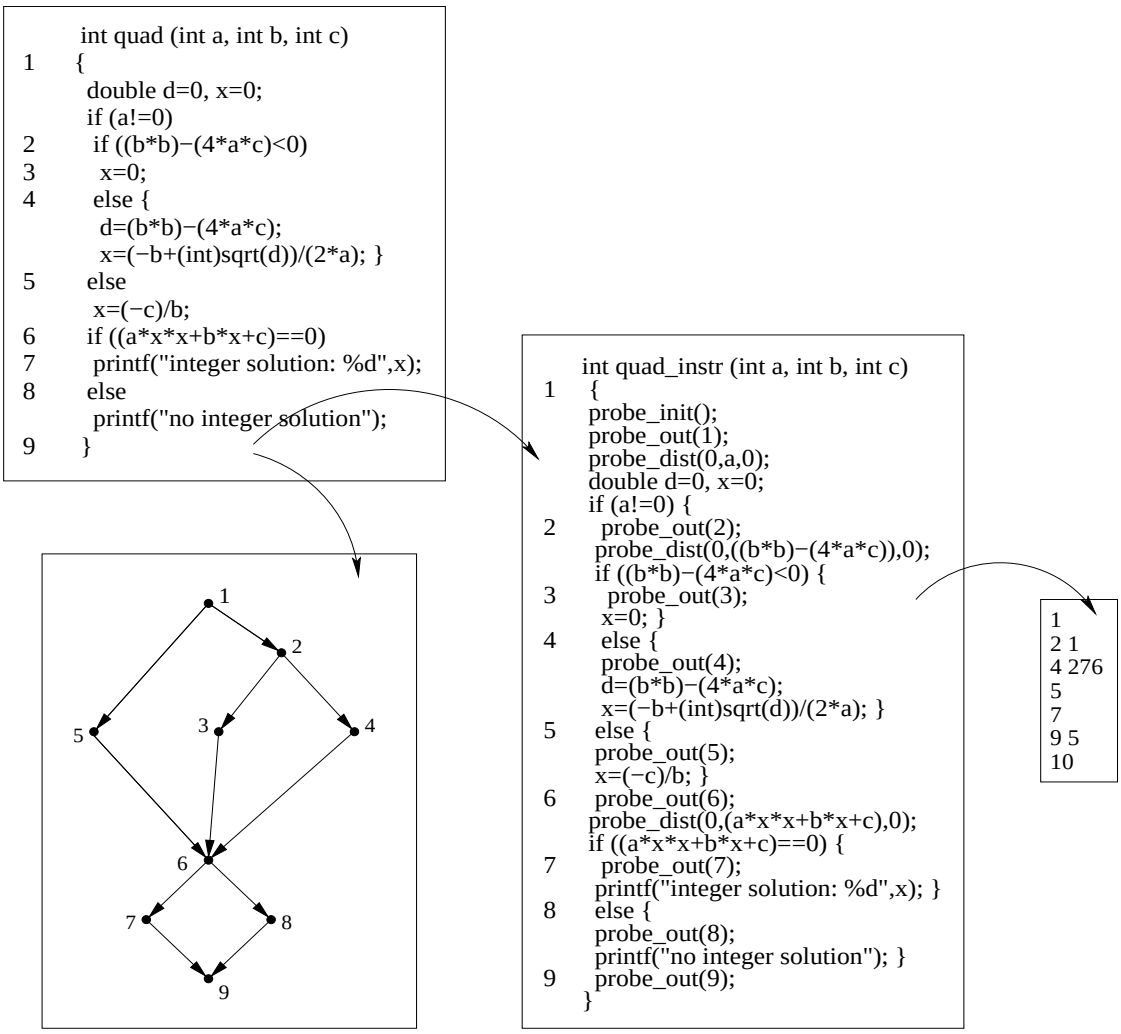


Figure 4.3: Example of source code, control flow graph, instrumented version, and output information.

After an execution of the instrumented code, an output file contains, at each line, the traversed basic block and, if the previous block had a conditional statement, the values of the (sub)expression(s) in it calculated according to equation 3.1. As seen in the figure, the instrumentation uses three artificially created probe functions. At the beginning, the call to *probe_ init* initialises the output file and a required memory vector. The *probe_ dist* function calculates an appropriate distance measurement between the second and third parameters and stores the result in the position of the memory vector given by the first parameter. In this example, the distance will always be the absolute value of the difference between the parameters, as they are numerical. The *probe_ out* function writes, in the output file, a new line containing the basic block number given as a parameter and, if any, every (sub)expression value in the memory vector. An example of an output file is also presented in Figure 4.3.

In the system startup, program branches are detected from the control flow graph and bound with a population of individuals created by sampling a uniform distribution.

Once this is done, the test case generation iterative process begins. Assuming that the branch defined by arc $(2,3)$ is selected as the objective in a round, an EDA will pursue its coverage. Following equation 3.1, with $K = 1$, the objective function is

$$f(\boldsymbol{x}) = \begin{cases} M & \text{if block 2 not reached} \\ |(b^2 - 4 \cdot a \cdot c) - 0| + 1 & \text{if block 2 reached and block 3 not reached} \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{x} = (a, b, c)$ and M is the highest computable value.

Representing each integer-valued parameter with 16 bits, an individual will be a 0-1 string of length 48. For its evaluation, this string is translated into a program input which will be given to the instrumented program for execution. Supposing that the input obtained is $(1, 20, 31)$, then the instrumentation results are those of Figure 4.3. The fitness of the individual will be the value of f for the input. As the exercised branch was $(2, 4)$, in the line of block 4 the result of $|(b^2 - 4 \cdot a \cdot c) - 0|$ is given, so $f(1, 20, 31) = 276 + 1 = 277$. Although the individual is already evaluated, the performed execution is not discarded and the output file is used to calculate the function value for every branch different from the objective. If the value improves the fitness of the worst individual in the set associated with a branch, then this worst individual is replaced by the actual one and the branch is marked as untreated.

The EDA will run for branch $(2, 3)$ until a maximum number of generations is reached or the minimum of f is found. In the first case, the branch is marked as treated but uncovered and in the latter as covered. Once the EDA finishes, another iteration of the overall process begins.

4.5 Experimental Evaluation

In order to observe how this approach performs in practice, several experiments were carried out. A handful of EDAs overviewed in Section 2.3 are considered to generate test inputs for a number of programs extracted from the literature [142; 233; 246]. The purpose of the evaluation is twofold: analyzing the performance of the approach with different EDAs and comparing their results with those attained by other alternatives.

4.5.1 Experimental Setting

The experiments involved seven classical programs which are commonly used for validation in the field. Although most of these programs implement relatively simple algorithms, their source codes include a number of challenging branches for a test data generator. Anyhow, difficultness of branch coverage depends on the source code, so the implementations used here were those employed for experimentation in other works. Programs are outlined next.

`ClassifyTriangle`

This is a popular program in software testing experimentation. An input is composed of three numerical parameters, each representing the length of a segment. The aim is to detect the triangle type, if any, associated with the input. Four different versions were used. The `Triangle1` program [246] has three integers as input parameters, which in the experiments took values in the interval $[-16384, 16383]$. `Triangle2` [246] is the same as `Triangle1` with floating point parameters instead; the interval for each was $[-98304, 98304]$. On the other hand, `Triangle3` [142] is a new implementation where the parameters are integers for which the interval $[-512, 511]$ was chosen. Finally, `Triangle4` [233] constitutes a distinct implementation once again; the selected interval for its integer-valued parameters was $[-512, 511]$.

`Atof`

Given a string of characters as input, `Atof` [246] transforms it into a floating point number if possible. For the experiments, the input string length was 10 characters codified with 7 bits each (the ASCII character set).

`Remainder`

This function [233] calculates the remainder of the division of two integers. Therefore, an input is composed of two interger-valued parameters for which the interval $[-32768, 32767]$ was chosen during experimentation.

`Complexbranch`

In this case, there is no specific functionality as it is a function artificially created for testing purposes [246]. Its main characteristic is the existence of several hard to cover branches in the code. Six integer-valued parameters form an input taking values in the interval $[-512, 511]$.

In order to have an idea about the programs characteristics, Table 4.1 reflects the values for several structural complexity measurements [226]. The branch number column shows the number of branches, i.e. the number of optimization problems to be solved. Branch nesting depth points out the maximum nesting level for the branches in the code, that is, the maximum number of conditional statements that must be adequately fulfilled to cover a branch. In Myers interval, the lower bound is the number of conditional statements plus 1 and the upper bound is the number of expressions of conditional statements plus 1, thus referring to the compound expressions.

These measurements give a clue about the intricacy level of the structure of the source code. However, care must be taken of misinterpreting them as they do not necessarily represent the complexity of the program with regard to branch coverage. According to the approach here exposed, the complexity of an objective branch attainment will be defined by two sources:

- the difficulty in reaching the associated conditional statement; if it is hard, then many individuals will take the value M as fitness and the landscape will be plateau shaped,
- the function determined by the distance when the conditional statement is achieved.

The experiments involved several EDAs described in Section 2.3, namely: UMDA, PBIL, MIMIC, TREE, $\text{EBNA}_{\text{K2+pen}}$ and EBNA_{BIC}. That is, two EDAs from each of the three types described in Section 2.3 were chosen. For the PBIL algorithm, the value of the α parameter was set at 0.5. For each EDA and each program, four different population sizes as well as four values for the maximum number of generations were considered. Notice that the evaluation of an individual implies the execution of a program, which may turn

Program name	Branch number	Branch nesting depth	Myers interval
`Triangle1`	26	7	(14 : 21)
`Triangle2`	26	7	(14 : 21)
`Triangle3`	20	6	(11 : 18)
`Triangle4`	26	12	(14 : 14)
`Atof`	30	13	(16 : 41)
`Remainder`	18	5	(10 : 10)
`Complexbranch`	22	5	(11 : 23)

Table 4.1: Characteristics of experimental programs.

pop.size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	3965 96.92	3205 98.08	3625 97.69	2945 98.46	3700 96.92	3960 96.15*	5145 97.69	5480 97.31	4460 98.08	5035 97.31	7545 96.54	6665 96.92
100	6900 97.31	6530 98.08	5210 98.46	4360 99.23	6330 97.69	5540 98.08	6380 99.23	7920 98.85	7060 98.85	7100 98.46	13110 96.15*	10080 97.31
200	10900 98.46	13120 98.08	10300 99.62	8660 99.62	10160 98.08	6120 99.62	10300 99.62	13000 100	10720 99.62	8860 100	7120 100	9620 99.23
400	21640 99.62	25320 99.23	18120 99.23	16600 99.62	13480 100	12000 99.62	21160 100	30560 99.62	19400 100	14520 100	13360 100	14840 99.62
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	6535 98.46	8250 97.69	7055 99.23	7755 98.08	11690 96.92	10765 96.92	8005 98.46	14135 96.92	5675 99.62	10475 97.69	19090 96.15*	22115 96.15*
100	16050 97.69	10580 99.23	6150 100	8120 99.23	15430 98.08	16560 97.31	8000 99.62	13790 98.85	11130 99.23	15530 98.46	21340 97.69	20720 98.08
200	16640 99.23	12840 100	16640 99.23	8900 100	14200 99.23	17200 98.85	14680 99.62	20380 99.62	8820 100	14260 99.62	13160 99.62	6560 100
400	21080 100	30440 100	20840 100	16320 100	21720 99.62	17960 99.62	31960 99.62	27960 100	21400 100	17680 100	25320 99.62	22920 99.62

Table 4.2: Results for `Triangle1`.

the test data generation in a computationally expensive process. Thus, we restricted to ten executions of the generator for each combination of the parameters.

Within an EDA, at each generation, half of the population was selected according to a rank-based strategy. New individuals were simulated from the learnt probability distribution by means of Probabilistic Logic Sampling [103], and the population was created in an elitist way.

In Tables 4.2 to 4.8, the results from the experiments are shown. In each cell, the average results from the ten executions are provided. The first row provides the average number of generated test inputs during the process and the following is the average coverage measurement. If the highest coverage achieved in the ten executions is not 100%, then this value is labelled with an asterisk.

pop.size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	3315	4215	3225	4920	5145	4305	7365	7620	5565	7385	6215	8885
	96.92	96.54	98.08	97.69	95.77	96.15	96.92	97.31	98.46	96.15	97.69	98.85
100	7330	6330	5730	4420	9860	9020	9120	9980	10430	6980	13150	18120
	97.69	98.46	97.69	98.46	96.54	95.39	98.46	98.08	98.08	98.85	97.31	96.54
200	8840	9780	7360	8220	9660	8320	10940	6200	16300	19920	19980	8400
	98.85	98.85	99.23	99.23	97.69	98.85	99.23	100	98.85	97.69	97.69	99.23
400	12440	15720	16520	11680	11760	10920	211160	15160	13680	18160	16680	9360
	100	100	99.62	99.62	99.23	100	100	100	99.62	99.23	99.23	100
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	4085	7960	8555	10225	13825	17230	149755	25495	12795	10715	11385	16770
	99.23	99.23	98.85	98.85	98.85	97.69	97.69	96.92	98.08	98.85	99.23	99.62
100	11080	14220	11510	16790	20360	21520	15470	21210	20600	17820	33980	23150
	98.46	98.46	98.85	98.46	98.46	98.08	99.23	98.08	98.85	99.23	98.85	98.46
200	14920	23280	17960	17500	20820	12960	29060	13820	35100	32800	49000	23940
	100	99.23	99.23	99.62	99.23	99.23	98.85	99.62	98.46	98.85	97.69	99.23
400	10640	14000	33400	12920	33960	26080	16280	12880	10800	21840	55320	22560
	100	100	99.85	100	99.23	99.62	100	100	100	99.62	98.85	99.62

Table 4.3: Results for `Triangle2`.

pop. size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	3150	2950	2960	3700	4485	4450	3090	3570	3780	4435	4990	7675
	98	98	99	96	94	94.5	99.5	98.5	98	97.5	97.5	95.5
100	5440	5780	5990	3820	6930	5640	4790	5780	6740	3880	7240	4730
	98.5	97.5	98	99.5	95	97.5	100	100	99	100	98	99
200	10500	9600	7820	8160	8300	8520	9300	11080	10340	8920	12940	10560
	99	97.5	100	99.5	98	99	100	100	99.5	100	98	100
400	17600	21960	18840	18520	15480	16240	19400	27200	22320	15960	15320	15320
	98.5	95.5	99.5	99.5	100	99.5	100	100	99.5	100	100	100
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	4540	4700	5965	4965	9585	11500	3875	6750	4805	6580	15920	16405
	99	99	99	99	98	98.5	100	98.5	99.5	99.5	98.5	98
100	5990	6220	7460	5330	7930	8010	6320	9240	8990	4080	12650	6660
	100	100	99.5	100	99.5	99.5	100	99.5	99.5	100	99	100
200	10160	13960	10600	8600	13920	21380	10640	10420	8460	8640	13500	7080
	100	100	100	100	99	98.5	100	100	100	100	100	100
400	18680	25000	21280	15440	21000	16440	20800	29600	21080	15200	21080	15800
	100	100	100	100	100	100	100	100	100	100	100	100

Table 4.4: Results for `Triangle3`.

pop. size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	21975 88,85*	20500 89,62*	26205 88,85*	10535 89,23*	12630 88,85*	12530 85,39*	30700 89,23*	26355 88,08*	28955 90,39*	18460 89,62*	24300 86,15*	24145 88,08*
100	47600 90,77*	48870 93,08*	45870 93,46*	24520 93,46	21140 90,39*	16970 90,77*	55660 91,54	48190 96,15	51010 93,85	30980 94,23	36930 90*	40630 90*
200	69820 97,31	96400 98,08	94980 94,23	37280 97,69	23500 95,39	17700 96,92	106960 96,54	99340 96,54	81320 96,92	31020 99,23	50560 93,46	28060 97,69
400	128120 96,54	119120 96,92	105400 97,31	43160 99,62	27960 99,23	22720 100	154000 98,08	158520 98,08	157480 98,46	45720 100	22760 100	48240 98,46
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	50030 89,62*	49755 87,69*	43105 91,15*	35835 91,15*	49445 86,92*	42035 86,92*	53630 93,46	57515 91,15*	47260 93,08*	47815 90,77*	67930 87,69*	72610 86,92*
100	84000 93,08	77980 95	80290 91,92	51270 93,08*	64230 90,39*	68200 90,39*	114520 92,69	113360 92,31*	79920 93,46	55230 94,23*	98600 90,77*	118550 89,23*
200	92220 97,69	107560 98,46	130800 95,39	43780 98,46	73680 93,85	67100 95,77	180200 96,15	151800 97,31	133480 96,92	52620 98,46	138280 93,85	132520 93,85*
400	147000 99,23	122040 99,62	168960 98,08	60840 99,23	50200 98,85	34560 100	225800 100	174880 98,85	141920 99,23	60800 100	86080 98,46	24080 100

Table 4.5: Results for `Triangle4`.

pop. size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	13330 87,33	12675 87	13525 89,33	14350 87,33	6600 97	6555 98,33	9640 98,33	34050 81,33*	12795 97	32230 83,67	7685 100	8850 98
100	28850 87	17740 94,67	28310 86,67	30080 87	20380 95	13840 97,67	36670 94,67	37050 91,67	18090 100	40360 92	21970 97	20770 98
200	73780 83	46180 92,67	63140 84,67	62760 87	59020 88,33	72000 84,33	139280 83	43400 98,67	92780 91,33	119920 88,33	50460 96,33	54820 96
400	165400 80,67*	164920 88	148760 88	157640 87,33	150720 80,67*	131320 91,33	193720 94	125040 97	146160 92,67	150480 95	158040 93,67	196560 94,33
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	9905 100	41790 88,67	9125 99,67	15665 99,33	11850 99	11015 99,67	17125 99,33	60995 88	13350 100	34315 96,67	14190 99	9505 100
100	31990 98,67	32340 97,67	22660 100	54370 94	28310 98,67	27800 99	32780 99,67	67940 94*	30250 100	59550 99,33	27560 99,33	31380 99,33
200	79920 98,33	89280 95	51400 100	104020 96,67	74760 97	68140 97,67	66100 99,33	119500 94,67	74020 99	115880 98	86260 98,67	65880 98,67
400	150520 98,67	105560 99	214480 96	259400 95,33	160160 98,33	147320 98,67	179920 98	180280 98	203200 100	203480 98,67	187520 98,67	221680 97,33

Table 4.6: Results for `Atof`.

pop. size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	2525	3275	2560	2360	3235	2370	6870	5985	4570	4745	6010	5510
	95.56	94.44	95.56	96.11	93.89	96.67	92.78*	93.89	95.56	95.56	93.89	94.44
100	5310	6700	6310	4810	5130	5790	8410	14620	11690	7010	6260	7860
	95	93.89	93.89	95.56	95.56	94.44	96.11	92.22	93.89	96.67	97.22	96.11
200	14140	13360	11340	4240	4920	6000	20600	20180	25500	12140	15920	2360
	92.78	93.89	94.44	98.89	98.33	97.78	95	95	93.33	97.22	96.11	100
400	22520	28280	20640	10160	8200	5840	46640	43000	39000	15680	4200	11080
	95	93.89	95	98.33	98.89	99.44	93.89	94.44	95	98.33	100	98.89
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	12460	9830	13350	10295	6700	10475	12715	18555	18410	12490	17025	16995
	93.33	95	92.78	95	96.67	94.44	95.56	93.89	93.33	95.56	94.44	93.89
100	15410	17370	20890	20730	24760	16870	48440	31430	22270	13110	30850	24850
	96.11	95.56	94.44	94.44	93.33	96.67	91.11*	94.44	96.11	97.78	95	95.56
200	26820	41920	26000	14360	18040	29540	73320	50860	49820	31820	25520	19740
	96.67	94.44	96.67	98.33	97.78	96.11	93.33	95.56	95.56	97.22	97.78	98.33
400	83760	62440	74600	19160	4000	3800	99640	136120	99240	15920	3920	4320
	94.44	96.11	95	98.89	100	100	95.56	93.89	95.56	99.44	100	100

Table 4.7: Results for `Remainder`.

pop. size	max. generations											
	50						100					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	8030	7145	7765	6885	6855	6710	10570	12085	9845	9515	8555	10140
	93.18	94.09	93.18*	94.55	92.73	92.27	95.46	95.46	96.82	95.46	96.36	95.46
100	12150	13570	13690	11490	10440	10320	14090	16070	15010	16510	11060	13120
	96.36	95	95.46	95	95	95.91	97.27	99.09	97.27	95.91	99.09	99.09
200	20600	19920	21500	18100	13640	14060	24780	20340	18000	21800	18800	18380
	97.73	96.82	95.91	97.73	97.27	97.73	96.82	98.64	99.09	97.73	98.18	98.64
400	24560	18680	24600	22200	23840	23360	27560	25240	27960	22960	22080	24240
	98.18	99.55	98.18	99.55	97.27	97.27	99.09	99.09	100	100	100	99.55
	200						300					
	UMDA	PBIL	MIMIC	TREE	BIC	K2	UMDA	PBIL	MIMIC	TREE	BIC	K2
50	19875	17815	15480	17425	9320	10390	12275	22855	18760	18540	15510	12915
	96.82	95.91	98.18	96.82	99.09	97.27	98.64	95	98.18	97.27	98.64	99.09
100	29050	20540	20070	16360	10860	11930	20920	22350	17130	21360	19400	17270
	98.18	97.73	98.64	98.18	99.55	100	99.55	98.64	100	100	99.09	100
200	26200	21340	30300	22740	20780	21720	22440	30300	20580	25820	26200	30340
	99.55	100	99.09	100	99.55	99.09	100	100	100	100	100	100
400	24040	25560	34360	39120	33880	31520	31320	42640	23960	27800	30400	33920
	100	100	100	100	100	100	100	99.55	100	100	100	100

Table 4.8: Results for `Complexbranch`.

4.5.2 EDAs Performance

Results presented by Tables 4.2 to 4.8 reveal that in 94% of the cases the average coverage surpasses 90%, and if the population is big enough, 100% is reached for every program. Although the tables do not show this, when the highest achieved coverage is not 100% (asterisk values) then the best execution obtained a value higher than 90% in all cases except three from the `Triangle4` program, which reached more than 88%.

The most difficult programs for the test case generator seem to be `Triangle4` and `Atof`. Table 4.1 reveals that `Triangle4` has no compound expression in its conditional statements although the nesting depth is one of the largest. In fact, several of the expressions are of the form $\mathcal{A} = \mathcal{B}$, which are usually the most difficult ones to fulfill. With regard to the `Atof` program, the nesting depth is the highest one and three-quarters of the expressions are compound. The comparison operator in many of the subexpressions are also equalities.

In order to observe the behavior of the objective selection phase in the generator, for each program, the number of times the search for an objective is repeated was recorded during the executions. The average is zero or almost zero for all the programs except `Triangle4` and `Atof`. In these two, `Triangle4` reaches higher values, with 7 as the maximum, and in `Atof`, the highest value is 1.8. In both cases, this value decreases as the maximum number of generations increases. In fact, in `Triangle4` it is near zero in most of the cases for 300 generations, and in `Atof` the value is zero or near zero for 100, 200 and 300 generations. Thus, according to these results and considering the set of individuals associated with a given objective, it seems that the number of times that an individual that improves the fitness of the set is found increases with the program complexity for branch coverage. Therefore, the number of optimization problems being solved during the process and, consequently, the number of generated inputs, becomes higher.

Regarding the optimization phase in the test case generator, Tables 4.2 to 4.8 also provide interesting information concerning the differences between EDAs. The coverage value is a main gauge of the performance of a test data generator. However, the number of inputs obtained reflects the effort made during the process. Therefore, it is important for a generator to obtain a coverage value with the lowest cost, that is, producing as few inputs as possible. This implies that, given two generators achieving the same coverage, the one yielding the fewest inputs is preferred. So, considering full coverage a mandatory requirement, multivariate EDAs offer the best results as they create the lowest or second lowest number of inputs in all the programs except in `Triangle3`. Taking the ratio between generated inputs and achieved coverage into account, the best values are shared by bivariate and multivariate EDAs. Precisely, $\text{EBNA}_{\text{K2+pen}}$ has the best ratio in three of the programs, and TREE and MIMIC in two of the programs each. These best ratios belong to the cases of 50 individuals and 50 generations, with the exception of `Remainder`, which obtained the best ratio with 200 individuals and 100 generations.

Program name	UMDA	PBIL	MIMIC	TREE	BIC	K2
`Triangle1`	32	39	27	**21**	30	25
`Triangle2`	**17**	18	19	20	20	**17**
`Triangle3`	19	27	20	**16**	24	19
`Triangle4`	32	31	31	**19**	20	20
`Atof`	20	24	18	26	**16**	**16**
`Remainder`	25	27	25	18	17	**16**
`Complexbranch`	19	20	19	17	**16**	**16**
Total	164	186	159	137	143	**129**

Table 4.9: Rank of EDAs with regard to the number of generated inputs.

In general, EBNAs obtain the worst results when the population is small, in which case univariate EDAs become competitive. However, as population size grows, multivariate EDAs improve their average coverage and, when 100% is reached, the number of generated inputs is usually lower than in the rest of EDAs. This indicates that, when adequate parameter values are met, EBNAs obtain the optimum in fewer generations than the other alternatives. These results reinforce the suggestion made in Section 4.3.1 about the adequacy of multivariate EDAs when dealing with this problem.

In order to statistically validate these concepts, two rankings based on hypothesis tests were carried out, one over the number of generated inputs and the other over the coverage measurement. For each program and for each value of population size and maximum number of generations, EDAs were ranked as follows. First, EDAs are ordered according to their average value in the result being considered, i.e. number of generated inputs (increasing order) or coverage (decreasing order). If a tie occurs, the involved EDAs are ordered by their variance. Then, following the order obtained, several Mann-Whitney tests are performed, each of which designates a rank value to an EDA. The first sample in the test is formed by the data from the i-th EDA in the order, and the second sample is the data from the $(i+1)$-th EDA. If the test finds significative differences at a 0.05 confidence level, then the rank value of the i-th EDA plus 1 is given to the $(i+1)$-th EDA. Otherwise, this EDA is designated with the same rank value as the i-th EDA and the first sample of the next test is extended with the data from the $(i+1)$-th EDA. Once a rank is obtained for all the different combinations of population size and maximum number of generations, the sum of the rank values of each EDA is calculated.

Tables 4.9 and 4.10 show the ranks for the number of generated inputs and coverage measurement respectively. The last row of each table provides, for each EDA, the sum of the ranks in all the programs. The best values are marked in bold. As can be observed, with regard to the number of inputs, multivariate EDAs take a larger number of the best values than do other types of EDAs. Considering the last row, the best EDA is

Program name	UMDA	PBIL	MIMIC	TREE	BIC	K2
`Triangle1`	**16**	17	**16**	**16**	18	18
`Triangle2`	**16**	**16**	**16**	17	17	**16**
`Triangle3`	**16**	17	**16**	**16**	20	**16**
`Triangle4`	20	19	20	**16**	26	26
`Atof`	22	24	18	22	19	**17**
`Remainder`	24	23	22	17	17	**16**
`Complexbranch`	17	18	**16**	18	**16**	**16**
Total	131	134	124	**122**	133	125

Table 4.10: Rank of EDAs with regard to the coverage measurement.

$EBNA_{K2+pen}$, followed by TREE and $EBNA_{BIC}$. The PBIL algorithm is the worst EDA, although it must be noted that its results heavily depend on the α parameter, as was shown in [89]. Taking the coverage measurement into account, the differences are not so clear. The total rank in the last row reveals that the values of the first three EDAs are quite similar and that there is a 12-unit difference between the first and the last EDA. In this case, TREE offers the lowest value, MIMIC is the second best EDA and $EBNA_{K2+pen}$, the third. Thus, it can be concluded that the best overall EDAs in the experiments, with regard to the coverage and generated inputs results, are TREE and $EBNA_{K2+pen}$.

4.5.3 Comparison with Other Works

The programs here considered for experimentation were extracted from previous works in the literature: `Triangle1`, `Triangle2` `Atof` and `Complexbranch` from [246], `Triangle3` from [142], and `Triangle4` and `Remainder` from [233]. All these works use a GA as the optimization technique. Next, the results obtained by such GAs based approaches are faced to those of the EDAs based test data generator.

When comparing the results from Tables 4.2 to 4.8 with those of their respective works, it must be noted that the plain form of EDAs was applied in the experiments, while in the referred works, sophisticated forms of GAs are used. In [246], a coarse grained parallel GA is chosen, in [233], different genetic operators and parameters are considered, and in [142], a simple GA and a differential GA are employed.

Besides the Evolutionary Algorithm, two other aspects must be kept in mind during the comparison. On the one hand, the fitness function in the EDA based approach differs from the one in two of the works. In [246], fitness is calculated from a function of the type described in equation 3.2. In the case of Sthamer [233], a function of the type in

Approach	Triangle1	Triangle2	Triangle3	Triangle4	Atof	Remainder	Complexrbranch
Other work	16915	42086	-	27876	35263	644	28978
EDA based	6150	6200	3875	22720	7685	2360	11930
Proportion	36%	15%	-	82%	22%	366%	41%

Table 4.11: Number of inputs generated by the EDA based approach and other approaches.

equation 3.1 is presented taking two distances for each program into account: one is the absolute value of the difference between the numerical representations of the operands (as in here), and the other is the Hamming distance between the binary representations. On the other hand, the interval of values taken by the input parameters also has to be considered. Neither in [246] nor in [142] are the intervals used in the experiments clear. In [233], for each program, the results with several intervals are presented.

Fixing the coverage at 100% and taking the best results reaching this value, Table 4.11 shows the number of inputs generated by the EDA based approach and the other works. The last row provides the percentage of inputs of the EDA based approach with regard to the other work.

In the case of `Triangle3` no value is included, since the number of generated inputs is not revealed in the results presented in [142], and the average coverage attained in the five performed executions is 93%. However, it must be taken into account that this work deals with condition/decision coverage, which is a criterion subsuming branch coverage.

Apropos of [246], considering the input parameter intervals used in the experiments, the EDA based approach generated less than half the inputs in this other work for all the programs.

The best results in [233] for the `Triangle4` were obtained with the $[-100, 100]$ parameter interval for the inputs and the distance based on the absolute value of differences. Regarding `Remainder`, however, the outstanding results corresponded to the $[-20000, 20000]$ parameter interval and the Hamming distance, Therefore, the values in Table 4.11 correspond to these configurations. For `Triangle4`, aside from the improvement shown in the table, the number of inputs generated by the EDA based approach is 88% of the inputs in [233] for the Hamming distance. However, in the `Remainder` program, the results are not outperformed, neither for the Hamming distance (as the table shows), nor for the distance based on the absolute value of differences. In this last distance, the number of inputs generated by the EDA based approach was 250% of the inputs in the work by Sthamer.

The results of these experiments conform, in general terms, to those in the application of EDAs to other problems [111]. Although in EDAs the quality of the solution is similar to the one achieved by GAs, the number of generations required and, hence, individuals generated is remarkably lower.

4.6 Summary

In this chapter, we have described an approach for the application of EDAs to the test data generation problem in the context of branch coverage. Several EDAs comprising different orders of dependencies in the probability distribution to be learnt were evaluated empirically.

Analyzing the results obtained from the experiments, a general conclusion can be drawn: EDAs prove to be a powerful option for tackling this problem. The coverage attained was 100% in all the experimental programs and the number of inputs generated was significantly lower than in other works, excepting a few cases. Among the different EDAs, algorithms using nontrivial probabilistic models seem to be a promising alternative. More precisely, TREE and $\text{EBNA}_{\text{K2+pen}}$ have shown the best overall performance. The capability of these EDAs for expressing the dependencies between problem variables could be a key point, as such dependencies usually exist when trying to cover a particular branch.

5 Software Test Data Generation by means of SS

The use of EDAs for test data generation in the previous chapter supports the application of other modern metaheuristics for solving this problem. While EDAs rely on a typically stochastic strategy, SS is conceived as a more systematic optimization method. In this chapter, the suitability of SS to generate tests inputs for branch coverage is studied. Additionally, EDAs and SS are combined in a collaborative scheme that aims at profiting from the benefits of both methods.

The chapter is arranged as follows. Once motivated, the SS approach is described, together with different alternatives for using the improvement method in the SS algorithm. Then, the results of the conducted experiments are analyzed. Appealing conclusions on the performance of the SS approach and the role of the improvement method in this context are obtained. In the second half of the chapter, the combination of EDAs and SS is explained and evaluated through experiments. Finally, EDAs, SS and their combination are compared to identify the best method.

5.1 Motivation

As pointed out in Chapter 3, when dealing with the generation of a test input covering a branch, the associated search space is usually large and complex. A well-known conjecture in Operations Research is that an appropriate management of the diversification and intensification concepts during the search in such spaces yields good solutions. These are the principles on which SS is based. This, together with the flexibility of the SS methodology, make it worthy of consideration for solving the test data generation problem.

In fact, SS has already been applied to several difficult optimization problems [33]. It has been compared with GAs in permutation problems [141], producing high quality solutions in fewer evaluations than GAs. Moreover, in [126], Laguna and Martí presented several SS designs to solve a set of nonlinear function minimization problems, obtaining encouraging results. Since the test data generation can be tackled as the resolution of a number of optimization problems, SS seems to be a promising technique to be studied.

In the work by Laguna and Martí [126], however, no improvement method was employed in the algorithms proposed. Thus, in order to shed some light on the SS methodology internals, we investigate the influence of this optional component in our context.

Moreover, EDAs have been applied in the previous chapter with encouraging results, so they constitute an adequate benchmark for comparison. This represents an opportunity for facing a stochastic optimization technique, like EDAs are, with a more deterministically oriented method, namely SS, in the context of a real-world problem. Furthermore, EDAs become a promising option from which SS may benefit in order to improve its performance.

So, the approach exposed here deals with inputs generation for branch coverage making use of the previous optimization methods in two different ways. Firstly, pure SS alternatives are presented in order to evaluate their performance and compare them with the EDAs based approach. Secondly, EDAs and SS are combined in a collaborative strategy.

5.2 The SS Approach

Similar to the EDAs test data generator, the SS alternative proposed here is based on the general scheme in Figure 3.5. In fact, the only algorithmical difference between both approaches concerns the optimization step. Here, the optimization problem associated with the objective branch is tackled by means of an SS algorithm, instead of by an EDA.

An input is represented as in the EDA approach, that is, as a 0-1 string, and the objective function value is obtained according to equation 3.1 as well.

The set of solutions associated to an objective branch, which has the same size as the set of diverse solutions P, is overturned in P at the beginning of each SS execution. Thus, in practice, the set of solutions can be viewed as a particular initial P set for each branch.

At the test data generation process start-up, the set of solutions of each branch b is created by introducing distinct solutions obtained via diversification and, if such is the case, via improvement. The evaluation of a solution is not only performed in relation to b, but for any other branch b' with no completely constructed set. If the evaluated solution outperforms the worst in the set of b' and the solution is not in the set yet, then it is introduced. It is important to note that, in case an improvement method is used, only the best solution found for b' is considered for inclusion in its set, thus avoiding the introduction of solutions coming from the same seed. When starting the creation of the set of solutions for b, a number of them may already be in the set. However, they are not improved with regard to b because they were found during the improvement of another branch, so the improvement method is applied to these solutions. Nonetheless, no matter how many solutions are already in the set, the inclusion of half of them via

diversification (and improvement) is forced in order to guarantee a degree of diversity in the set.

As may be noticed, once the process start-up finishes, every solution in the set of a branch is improved with regard to it. In order to maintain this property during test data generation, when a solution is evaluated and is to be included in the set of solutions of the branch, the improvement method is applied before entering the set. This way, at every moment the solutions in the set of a branch are improved with regard to it.

The SS stopping criterion consists of finding the minimum (covering the objective branch) or reaching a maximum number of iterations. If the current iteration is not the last and no new solution was added to $RefSet$, a rebuilding step is carried out. To be precise, a new set P is created and half the worst solutions in $RefSet$ are replaced by the solutions in P that increase the most the diversity in $RefSet$. We measure the diversity of a solution $\boldsymbol{x} \in \{0,1\}^n$ as $\min\{d(\boldsymbol{x}, \boldsymbol{x'}) \mid \boldsymbol{x'}$ is a solution in the current reference set$\}$, where $d(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{i=1}^{n} |x_i - x'_i|$.

In [126], none of the SS designs used by Laguna and Martí applied an improvement method, and this might be an important element during the search. Thus, in the present approach, with the purpose of shedding some light on how the use of the improvement affects the optimization process, the following options are given:

Improve After The classical way of improving the solutions, that is, after diversification or combination (Figure 2.8).

Improve Before An alternative consisting of using the improvement method just before entering $RefSet$. Once a solution has been created via diversification, it is included in P, and improvement is applied only if the solution is one of the $b_1 = |RefSet|/2$ high quality solutions used to construct $RefSet$. The remaining b_2 solutions of $RefSet$ are not improved since they are assumed to be diverse solutions. Notice that in the rebuilding step it is not necessary to improve the solutions from P. On the other hand, if a solution comes from the combination method, improvement is performed only if it is to enter $RefSet$.

No Improvement In this case no improvement is included in the SS algorithm.

Excepting those reaching the optimum, during an SS execution a number of solutions are obtained, improved and rejected if they do not gain entrance to $RefSet$. Therefore, the idea behind the *Improve Before* alternative is to reduce the number of generated solutions by restricting improvement to those entering $RefSet$. Taking such an idea further yields then the *No Improvement* option.

In order to complete the SS design description, the five methods needed to implement the algorithm are explained next.

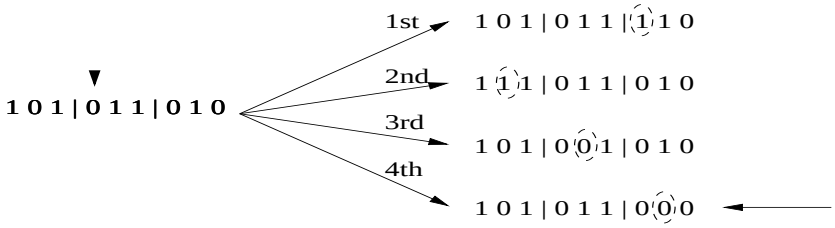


Figure 5.1: Example of local seach improvement method.

Diversification Generation Method

A simple implementation is adopted. Each solution is randomly generated according to a uniform distribution.

Improvement Method

The improvement method is a best first local search where the neighbors of a solution are to a Hamming distance of one. More precisely, the bit substrings codifying each input parameter are taken into account to define the order of evaluation of the neighbors. For a solution differing from the previous in the i-th bit of the substring codifying the j-th input parameter, the next neighbor to evaluate is obtained by changing the value of the most significant bit, previously unchanged, in the substring associated with the next parameter. To be exact, in the case of a parameter codification where the most significant bits are ordered from left to right, if the j-th parameter is not the last, the i-th bit of the substring belonging to the $(j+1)$-th parameter will be flipped. Otherwise, the $(i+1)$-th bit of the substring of the first parameter is changed.

Figure 5.1 presents an example of the local search method. The solution to the left is the initial solution codifying an input with three parameters; a vertical line divides the substrings representing each parameter. It is supposed that this initial solution was obtained by changing the signed bit. The neighbors considered in a hypothetical search step are shown to the right. The horizontal arrow indicates the assumed new best solution chosen for the next step.

Reference Set Update Method

The reference set updating follows a static update strategy. New solutions obtained via combination are placed in a pool. Once the pool is full, $RefSet$ is formed by the highest quality solutions already in it and the pool.

Improvement	Triangle1	Triangle2	Triangle3	Triangle4	Atof	Remainder	Complexbranch
After	10356	28761	31534	32646	1520023	145	30835
	100	100	100	100	82.33	100	100
Before	1575	2661	8325	10267	27087	240	9105
	100	98.46	99	100	67.33	100	93.18
No Local	2374	7561	3196	2549	27686	1235	5024
	96.54	97.31	95	98.08	64.67	97.78	94.55

Table 5.1: Experimental results of the SS approach.

Subset Generation Method

All two-solutions subsets are created. Obviously, only the solution pairs not previously generated are taken into account.

Solution Combination Method

For each pair of solutions, their input representations $\boldsymbol{x} \in \Omega$ and $\boldsymbol{x'} \in \Omega$ are obtained, and four new solutions are created from the following linear combinations:

$$\boldsymbol{x_1} = \boldsymbol{x} + d \tag{5.1}$$

$$\boldsymbol{x_2} = \boldsymbol{x} - d \tag{5.2}$$

$$\boldsymbol{x_3} = \boldsymbol{x'} + d \tag{5.3}$$

$$\boldsymbol{x_4} = \boldsymbol{x'} - d \tag{5.4}$$

where $d = |\boldsymbol{x} - \boldsymbol{x'}|/2$.

5.3 Performance Evaluation of Scatter Search Designs

In order to observe how the SS approach performs in practice, several experiments were carried out. Test data was generated for all the programs used to evaluate the EDAs based approach, taking the same intervals of values for each input parameter (see Section 4.5.1).

After preliminary experimentation, the maximum number of generations for the SS was set at 10, the size of set P was 100 and the reference set size was 10. The results of the experiments for the three improvement strategies are shown in Table 5.1. For each improvement strategy and each program, the average values in ten executions are provided. The first row is the average number of generated test inputs during the process, and the following is the average coverage measurement.

It can be seen in Table 5.1 that when the classical improvement strategy (*Improve After*) is adopted, the attained coverage is equal or larger than in the two other cases.

Nonetheless, *Improve Before* offers good results, as it generates a considerably lower number of inputs (solutions) than *Improve After* while keeping the same or almost the same coverage, excepting `Atof` and `Complexbranch` programs. On the other hand, in the *No Improvement* option, coverage is generally lower than in the two other strategies; in fact, full coverage is reached for no program. Considering the number of inputs, as it could be expected, *No Improvement* obtains in general a lower value than *Improve After*. However, this is not held with regard to *Improve Before*, which generates less inputs than *No Improvement* in four programs.

Statistical tests were conducted to check whether significant differences exist among these results. Comparing Improve After and *Improve Before*, the Mann-Whitney test revealed differences at a 0.05 confidence interval for the coverage in `Triangle2`, `Atof` and `Complexbranch`, and for the number of inputs in all the programs. Facing *Improve After* with *No Improvement*, dissimilarities were stastically significant for the coverage and the number of inputs in every program. Finally, in *Improve Before* versus *No Improvement*, differences were observed for the coverage in all the programs but `Triangle2` and `Complexbranch`, and for the number inputs in every case but `Triangle3` and `Atof`.

Thus, these results indicate that, although the improvement method is an optional element of the SS methodology, its relevance is high. The lack of improvement hinders the SS design from attaining the best performance. Indeed, the number of solutions an SS algorithm generates during the optimization process depends to a great extent on the way improvement is applied. More precisely, in our context, the *Improve After* strategy attains the highest quality solutions, however, the less intensive *Improve Before* option may reach the optimum in some cases, generating less solutions. So, according to the outcomes from the statistical tests, taking coverage as a primary factor and the number of inputs as a secondary one, we may conclude that *Improve After* obtains the best performance for `Triangle2`, `Atof`, `Remainder` and `Complexbranch`, while *Improve Before* is the best for `Triangle1`, `Triangle3` and `Triangle4`.

An interesting aspect which may be useful when considering an SS algorithm is the influence of each method during the search. This can be seen in Figures 5.2 and 5.3 in the context of test data generation.

Figure 5.2 shows the coverage attained by the SS methods for the *Improve After*, *Improve Before* and *No Improvement* strategies. Specifically, the results of the diversification (divers), improvement (improv) - if such is the case - and combination (comb) methods are presented, together with the coverage obtained when evaluating a solution with regard to other objectives (eval). In the *Improve After* alternative, almost all the coverage is reached through improvement and evaluation of other objectives. However, in the *Improve Before* option, where improvement is not so intense, the coverage of the local search decreases and the weight of diversification increases. This also holds for *No Improvement*, but completely eliminating the role of the local search.

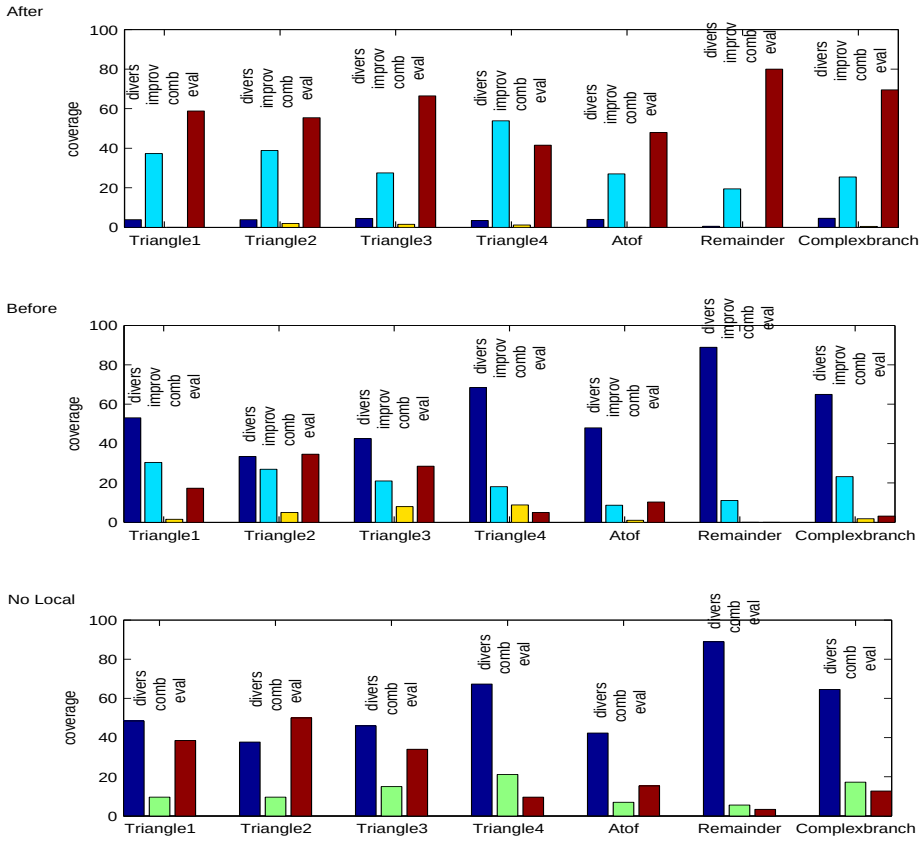


Figure 5.2: Coverage of SS methods for *Improve After* (above), *Improve Before* (middle) and *No Improvement* (below).

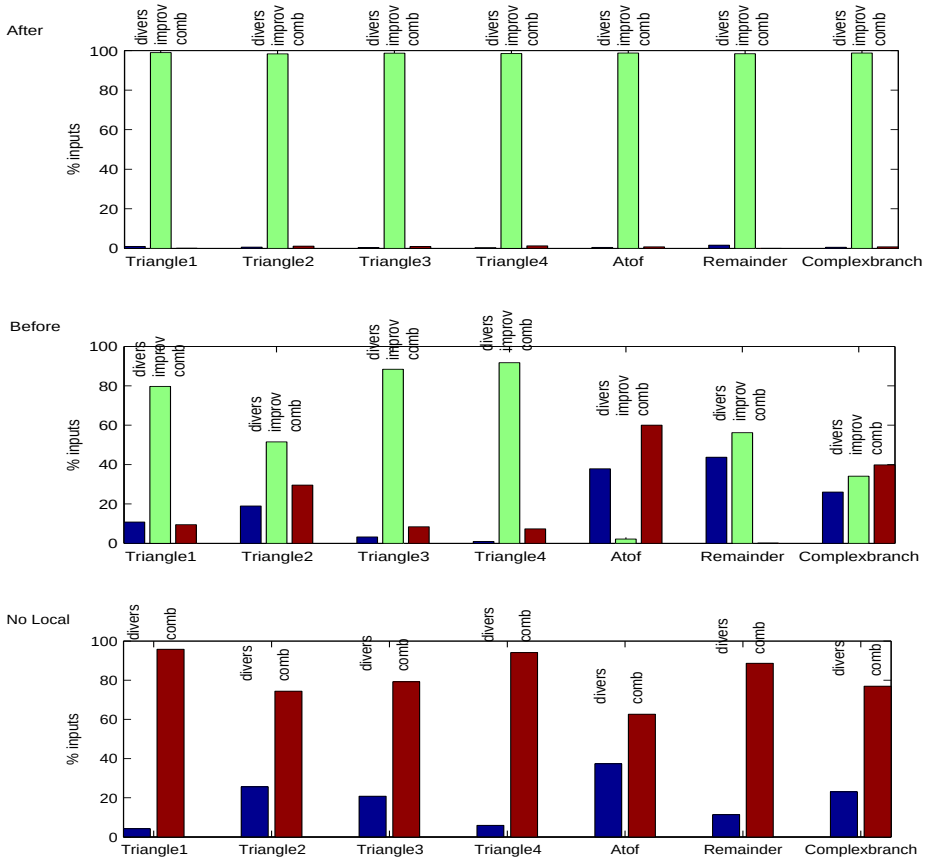


Figure 5.3: Proportion of inputs generated by SS methods for *Improve After* (above), *Improve Before* (middle) and *No Improvement* (below).

Approach	Triangle1	Triangle2	Triangle3	Triangle4	Atof	Remainder	Complexbranch
SS_{best}	1575	28761	8325	10267	1520023	145	30835
	100	100	99	100	82.33	100	100
EDA_{best}	6150	6200	3875	22720	7685	2360	11930
	100	100	100	100	100	100	100

Table 5.2: Results of the best SS and EDA approaches.

Alternatively, Figure 5.3 reveals the proportion of inputs generated by each SS method. Notice that, in general and where applied, improvement generates most of the solutions during the search; this is especially clear in the *Improve After* option. An interesting point is the efficiency shown by diversification in the case of *Improve Before* and *No Improvement*, since it generates a relatively low amount of inputs while offering most of the coverage. Indeed, the opposite happens for the combination method in *No Improvement*, which generates almost every input and contributes a low coverage.

A clear conclusion derives from these results. When used in a classical way, the improvement method plays a main role during the search, as it significantly affects the number and quality of generated solutions. In contrast, if improvement is applied differently, the behavior of other SS methods changes, especially in connection with the achievement of high quality solutions. More exactly, in this case the diversity method becomes apparently the main source of optima achievement. Additionally, if no improvement is employed, significance of the combination method increases with regard to the attainment of high quality solutions. However, the number of solutions generated by this method is huge in comparison to diversification, so suggesting a poor efficiency.

5.3.1 Scatter Search versus Estimation of Distribution Algorithms

In the previous chapter, EDAs were applied to the test data generation problem offering promising results. Hence, the EDA approach may be regarded as an appropriate benchmark for comparison with the SS test data generator. For the comparison, the best EDA and SS approaches in each case are taken into account. These approaches are identified by giving preference to coverage, that is, the best approach is the one that achieves the highest coverage and, if there is a tie, the approach with the lowest number of generated inputs.

Table 5.2 presents the results. SS_{best} and EDA_{best} denote the best SS and EDA approaches respectively; the format is the same as in Table 5.1.

The Mann-Whitney test was conducted to validate the results. Differences were found at a 0.05 confidence interval for coverage in `Atof`, and for the number of inputs in all the programs but `Triangle3`. So, we may conclude that in three of the seven programs, SS significantly outperforms EDA. More precisely, two of these SS approaches correspond to the *Improve Before* strategy, which makes it an interesting option for test data generation.

In contrast, although the *Improve After* strategy equals the coverage of the EDAs (except `Atof`), the number of inputs generated is higher in all cases (except `Remainder`). This is specially clear for the `Atof` program, where SS_{best} offers a poor behavior compared to EDA_{best}.

5.4 Scatter Search and Estimation of Distribution Algorithms Collaboration

The results obtained in some of the experimental programs suggest that SS approaches can generate good solutions with a low number of evaluations. Indeed, these results conform to those obtained in [126] and [141], where SS reached high quality solutions in fewer evaluations than GAs. Nonetheless, here, as well as in those works, it has been shown that there are functions for which the SS approach does not offer a good performance.

On the other side, EDAs were successfully applied to the automatic generation of test data in the previous chapter. However, in EDAs, it is difficult to set an explicit control of the diversification and intensification balance. By contrast, in SS this can be performed in a direct way due to its flexibility.

These observations motivated the idea of combining both optimization techniques. Both SS and EDA based approaches aim at generating test data for a given program by themselves. However, they could be entirely used in order to deal with the same problem, thus leading to a collaborative approach which may profit from the benefits of SS and EDAs.

The proposed collaboration consists of an EDA-SS approach where each search method acts separately. In other words, the EDA based generator is used first and, once it has finished, the SS based generator is employed over the remaining uncovered branches. This way, the general scheme in Figure 3.5 is first applied with an EDA and, if it was not able to solve the complete problem, the scheme is repeated with SS.

An important feature of the implementation developed is that, after the execution of the EDA generator, the SS approach initializes the set of inputs of each uncovered branch with the set resulting from the EDA. Thus, the SS generator starts the search using the best solutions found by the EDA. Although this seeding could involve a lack of diversity, the SS can recover from it through the diversification method in the rebuilding step.

5.5 Performance Evaluation of the Collaborative Approach

The collaborative approach was applied to the previous experimental programs in order to observe its performance.

EDA	Triangle1	Triangle2	Triangle3	Triangle4	Atof	Remainder	Complexbranch
UMDA	4015	4250	6281 ‡	41912 ‡	570306	6202 ‡	24154
	100	100	100	100	91.33	100	100
TREE	3272	10210	3870	24495	795693	6532 ‡	34900
	100	100	100	100	86.67	100	100
$EBNA_{K2+pen}$	5185	7452	7369	45902 ‡	1081907	4377 ‡	31653
	100	100	100	99.62	82.33 †	100	100
UMDA	3311	4700	3927	41440 ‡	28278	3012	11298
	100	98.85	99.5	99.23	79.67 †	100	96.36 †
TREE	3776	4857	3439	19560	30529	2614	9428
	100	98.85	100	98.46 †	78.67 †	100	96.82 †
$EBNA_{K2+pen}$	3317	4860	3548	12554	33620	2197	9189
	100	98.08 †	99	98.46 †	71.67 †	100	95 †

Table 5.3: Results of EDA-SS approach with *Improve After* (above) and *Improve Before* (below).

Recall that in Chapter 2 EDAs were classified in three types, according to the order of dependencies among the variables in the probabilistic model. Following the ranking carried out in the previous chapter, the best EDAs from each type were used in the experiments, i.e. UMDA, TREE and $EBNA_{K2+pen}$. Within an EDA, half of the population was selected at each generation according to a rank-based strategy. New individuals were simulated from the learnt probability distribution by means of Probabilistic Logic Sampling, and the population was created in an elitist way. Population size was the same as the one for set P. Since the SS generator may be used after the EDA, the maximum number of generations was relaxed to 10. In fact, a few preliminary experiments were conducted and the best results corresponded to this value. In the EDA literature, other works that have obtained good results with low parameter values in the experiments can be found [111].

The *Improve After* option of the SS approach attained, in four of the programs, a higher coverage value than the *Improve Before* alternative. However, the latter clearly generated fewer inputs than the former. Therefore, the SS generator was evaluated taking both options into account. The *No Improvement* strategy was not included in the experiments as it offered, with statistical evidence, an inferior behavior in every program. The SS parameter values were the same as in Section 5.3: $|P| = 100$, $|RefSet| = 10$, and 10 iterations at most.

Table 5.3 shows the results of the experiments; once again, the format is the same as in previous tables. The best approach (with preference to coverage) for each program is marked in gray.

As can be seen, these results conform to the ones obtained in Table 5.1, since the *Improve After* strategy reaches, in general, a higher coverage than *Improve Before*. Instead, the

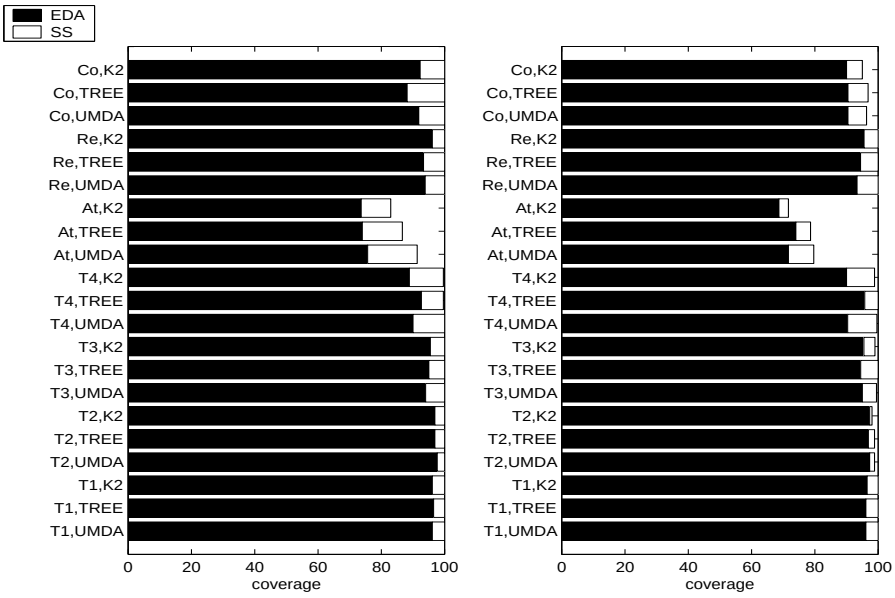


Figure 5.4: Proportion in the coverage of EDA-SS approach for *Improve After* (left) and *Improve Before* (right).

latter generates a lower number of inputs than the former.

Statistical tests were used to validate the best performance values. Since coverage is a primary measurement, for each program and each approach, the Mann-Whitney test was conducted with regard to the best coverage value (in gray). Then, for the cases where no difference was found, the test was again used over the best number of inputs generated. Table 5.3 presents the outcomes of the tests; symbol '†' denotes the cases where coverage dissimilarities (0.05 confidence interval) were found, while '‡' refers to the number of inputs.

In five programs, *Improve After* attains the best results. However, differences are statistically significant with regard to *Improve Before* in three of them: `Triangle4`, `Atof` and `Complexbranch` (and `Triangle2` just for $\text{EBNA}_{\text{K2+pen}}$). *Improve Before*, by constrast, offers a statistically sound improvement with regard to the other alternative only in `Remainder` (and in `Triangle3` for UMDA). Thus, *Improve After* shows a rough superiority to *Improve Before*.

In order to have an idea of the behavior of each generator in the collaborative scheme, Figures 5.4 and 5.5 present the proportion of coverage and generated inputs in the EDA and SS methods respectively.

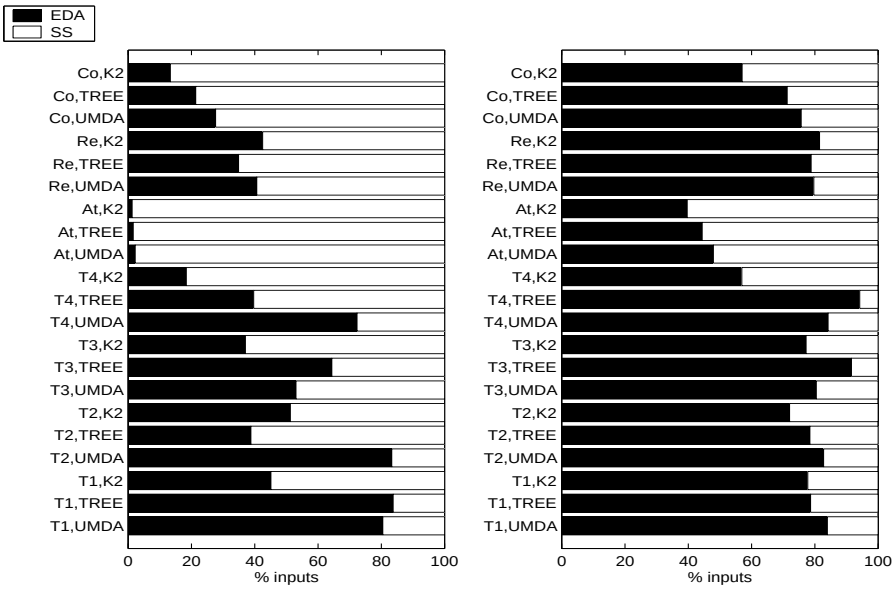


Figure 5.5: Proportion of inputs generated by the EDA-SS approach for *Improve After* (left) and *Improve Before* (right).

Approach	Triangle1	Triangle2	Triangle3	Triangle4	Atof	Remainder	Complexbranch
$EDA - SS_{best}$	3272 ‡	4250	3439	24495 ‡	570306	2197 ‡	24154 ‡
	100	100	100	100	91.33 †	100	100
SS_{best}	1575	28761 ‡	8325	10267	1520023	145	30835 ‡
	100	100	99	100	82.33 †	100	100
EDA_{best}	6150 ‡	6200 ‡	3875	22720 ‡	7685	2360 ‡	11930
	100	100	100	100	100	100	100

Table 5.4: Results of best EDA-SS, SS and EDA approaches.

Figure 5.4 reveals that the EDA based generator covers most of the objectives, since it operates first and can attain, among others, the easiest objectives. However, the EDA method is not able to reach a 100% coverage by itself, whilst this can be obtained by using the SS generator. In fact, the SS method always increases the coverage attained by the EDA based. Alternatively, Figure 5.5 shows how the use of improvement affects the results of the collaborative scheme, as the proportion of inputs generated by the SS with the *Improve After* alternative is higher than with *Improve Before*.

5.5.1 Collaborative Approach versus Others

The comparison of the EDA-SS approach with regard to the SS and EDA based test data generators can be observed in Table 5.4. In order to conform with the comparison in Table 5.2, the best EDA-SS collaboration ($EDA - SS_{best}$) is compared with the best SS (SS_{best}) and EDA (EDA_{best}) approaches. Once again, the best approach (with preference to coverage) for each program is marked in gray.

In order to validate the best values, the previous analysis with the Mann-Whitney test was applied here. So, analogously, Table 5.4 shows the outcomes from the tests; symbol '†' denotes the cases where coverage differences (0.05 confidence interval) were observed, while '‡' alludes to the number of inputs.

SS_{best} obtains the best values with statistical evidence in three programs, EDA_{best} in two, and $EDA - SS_{best}$ in one (no dissimilarity was found in `Triangle3`). It may be noticed, however, that excluding the programs where $EDA - SS_{best}$ is best, this approach generally lies between the two other approaches. Further statistical tests confirmed this at a 0.05 confidence level.

Similarly to Section 5.3, a poor performance is achieved in `Atof`. An explanation for this is that, even though preliminary experiments offered better results with the selected parameter values, in the case of `Atof` these values may not be appropriate. Nevertheless, Table 5.4 shows an increase in the coverage and a decrease in the inputs generated for $EDA - SS_{best}$ when compared to SS_{best}.

In any case, according to these results, the collaborative scheme may be considered a competitive alternative for test inputs generation.

5.6 Summary

This chapter has been devoted to the application of SS for solving the test data generation. The EDAs based approach is followed to fulfill the branch coverage criterion, though a SS algorithm is used instead of an EDA.

Three alternatives regarding the improvement method have been studied in this context. After experimental evaluation, it may be concluded that, despite being optional, the improvement method plays a main role in the SS methodology for this problem. The weight of improvement is reflected in the number of solutions generated (inputs) and the number of optima found during the search. Moreover, the way in which improvement is used in the algorithm affects the behavior of other SS methods. Following this idea, the *Improve Before* option proposed attained better results than Improve After in some programs, thus being an interesting alternative to the classical strategy. Clearly, the worst performance in the experiments is obtained if no improvement method is employed. In such a case, the combination method generates most of the solutions while reaching a relatively low number of optima by itself, so that the diversification method plays a main role in the search.

In addition, an EDA-SS collaborative scheme has been described to take advantage of the benefits of both methods. The experiments conducted on this alternative offered encouraging results. The collaborative approach offered the best or second best results in most of the test programs. However, in order to conclude this approach overcomes the isolated generators, future experiments have to confirm these results. Anyhow, the use of SS as a secondary optimization method improved the coverage of the previous EDA based method. Hence the collaborative strategy proves to be useful.

6 Enhancing the Test Data Generation Process: the Role of the Search Space and the Objective Function

When facing the test data generation as an optimization problem, two significant topics are the objective function and the search space. Although an active work is undergoing for the former, little attention has been paid to the selection of an appropriate search space. Hence, while making some hints on the influence of the objective function, this chapter concentrates on describing an alternative to the search space issue. More precisely, two approaches which employ an EDA as the metaheuristic technique are explained. In both cases, different regions are considered in the search for the test inputs. Moreover, in order to depart from a region close to the one containing the optimum, the definition of the initial search space incorporates static information extracted from the source code of the software under test. If this information is not enough to complete the definition, then a grid search method is used. According to the results of the experiments conducted, it is concluded that this is a promising option that can be used to enhance the test data generation process.

Remaining sections are organized as follows. After providing motivation, the benefits of an advanced form of objective function are empirically studied. Next, the alternative developed for selecting the search space is explained. Then, the experiments and the analysis of their results are shown.

6.1 Motivation

As we have seen throughout this dissertation, tackling the test data generation as the resolution of a set of optimization problems is offering promising results, and it constitutes nowadays a serious alternative to accomplish this task [143]. Nonetheless, depending on the program at hand, complexity in solving such optimization problems may be huge. In fact, in the context of branch testing, the search space defined by the inputs is often large and the objective function intricate, making the coverage of a branch a difficult task.

Most of the efforts to enhance the test data generation to date have concentrated on the optimization technique and the objective function. We have proposed in previous chapters modern metaheuristics that have shown encouraging results and present themselves as a promising alternative to more classical methods. By contrast, attempts on the objective function relate to the concepts in Section 3.5.2 for improving the function in equation 3.1.

Surprisingly, so far little attention has been paid to the selection of an appropriate search space. This is an interesting matter, as focusing the search on a promising region could simplify the problem, while making an inadequate choice an optimal solution (an input covering the branch) may not even exist.

An alternative regarding this question is suggested in [99]. There, a dependence analysis is applied to the variables in the source code to identify the input parameters that cannot affect the coverage of a given branch. This way, a number of problem variables can be eliminated and the search space reduced.

In the context of EAs, the search space matter can be tackled by Self-Adaptive Representation methods. These procedures may be classified as a form of parameter control [65] that, according to the behavior of the execution, dynamically transforms an individual's representation and, thus, the search space. Although it depends on the method, generally, the purpose of the transformation is to direct the search towards the most promising region found so far and avoid getting stuck in local optima [250].

This chapter is devoted to the enhancement of the test data generation by studying the influence of the objective function and, with more emphasis, by dealing with the search space. In both cases, we depart from the EDAs based approach presented in Chapter 4, so, from now on, in order to make the discussion more agile, this will be referred to as the *basic approach*.

Regarding the objective function, we will check whether an advanced design for the objective function, previously developed in [246], constitutes an improvement for the basic approach.

The bulk of the chapter concentrates on a novel alternative to the search space selection issue. The two major concepts which support this alternative are the use of *a-priori* knowledge on the problem instance to choose a search region, and modifying this region through the solution's representation. These concepts are applied to the basic approach. Initially, the EDA seeks for in a region chosen from the whole feasible search space. In order to select a promising region, its definition is based on static information extracted from the program's source code. In case this information is not useful to the definition, then a grid search method is applied. Additionally, during the process, the size of the region is increasingly widened. This way, if the objective entity is not exercised, a new search is performed on a larger region. This enlargement is applied to the approach from two points of view, giving rise to two algorithms.

6.2 The Influence of the Objective Function

Given the objective branch, finding the minimum of the corresponding objective function may be extremely hard owing to program semantics. As pointed out in Section 3.5.2, the function defined in equation 3.1 has some limitations that make the problem even harder. More precisely, a main drawback is the fact that every input not reaching the conditional statement associated to the branch receives the same function value, yielding a flat landscape. The function defined in equation 3.2 aims at alleviating this by returning a distance to the condition for such inputs.

To be exact, given a branch b and an input $\boldsymbol{x}$, let v_c denote the vertex representing the conditional statement associated to b in the control flow graph, and let p be the path from s to e (see Section 3.4.1) representing the flow of the program's control when executed with $\boldsymbol{x}$. We call a vertex v *control dependent* of a vertex w iff w represents a condition (i.e. $outdegree(w) > 1$) and there are both a path from w to v and a path from w to e not containing v. The condition distance d_c from a vertex $v \in p$ to v_c, $d_c(v, v_c)$, is defined then as the number of vertices in the path from v to v_c, on which v_c is control dependent. For the sake of convenience, in the particular case where no path exists from v to v_c, an infinite number of control dependent vertices is assumed, that is, $d_c(v, v_c) = \infty$. Equation 3.2 returns $min_{v \in p} d_c(v, v_c)$ for the inputs not reaching v_c.

One noticeable problem of this condition distance is the fact that several paths may exist from $v \in p$ to v_c. Thus, it could be the case of two paths contributing distinct values for $d_c(v, v_c)$. Such a situation is illustrated in Figure 6.1, where a source code segment and the corresponding portion of control flow graph are shown. In the figure, the two paths from v to v_c, i.e. $v, 2, 3, v_c$ and $v, 2, 5, 7, v_c$, own respectively 2 and 3 vertices on which v_c is control dependent. In [16], this problem was tackled through *optimistic* and *pessimistic* approaches, depending on whether $d_c(v, v_c)$ is taken as the minimum or maximum number of control dependent vertices respectively. Experiments under different scenarious were conducted for comparing both alternatives, however, no conclusion could be drawn about the superiority of either.

Since equation 3.2 augments granularity with regard to equation 3.1, it appears to be more adequate. Yet, to the best of the author's knowledge, no evaluation that checks this has been published. In order to compare both objective functions, we run the test data generator described in the basic approach using equation 3.2 and the best overall EDA from Chapter 4, i.e. TREE. All the experimental programs used there were chosen. For each program, the parameter values offering the best results (with priority to the coverage) for TREE were selected for execution. None of the programs shows the problematic situation in Figure 6.1, so there is no need to handle it.

Table 6.1 presents the results obtained with the classic (equation 3.1) and advanced (equation 3.2) functions. For each program, the table collects the average values in ten

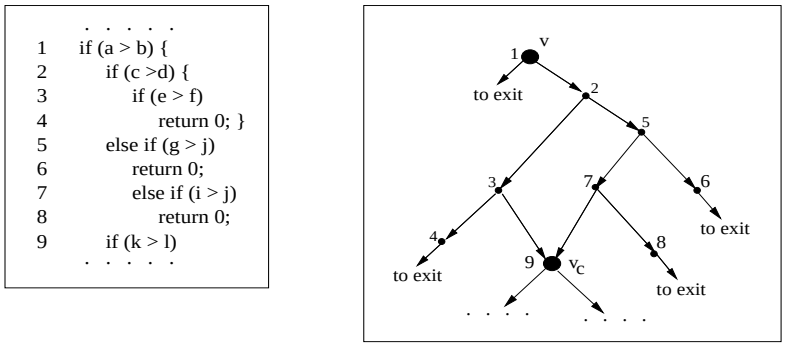


Figure 6.1: Example of two paths yielding different values for $d_c(v, v_c)$.

function	Triangle1		Triangle2		Triangle3		Triangle4		Atof		Remainder		Complexbranch	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
classic	100	8860	100	12920	100	3880	100	45720	99.33	15665	99.44	15920	100	21360
advanced	99.62	9940	100	5880	100	4180	99.62	51560	100	3475	100	1240	98.64	21420

Table 6.1: Results obtained with the classic and advanced objective functions.

executions for the coverage (%) and the number of inputs generated (#).

It can be noted that differences between both functions are slight for the coverage measurement and more prominent for the number of inputs in some cases. The Mann-Whitney statistical test was conducted over both measurements to validate the results. No significant dissimilarity was observed at a 0.05 confidence interval for the coverage. By contrast, differences were found for the number of inputs in `Triangle2`, `Atof` and `Remainder`, where the advanced function beats the classic.

Therefore, using the basic approach and according to these results, it may be concluded that the objective function from equation 3.2 improves or equals the performance of function from equation 3.1.

6.3 The Self-Adaptive Approach

If a test data generation system deals with this task as the process of solving a set of optimization problems, the search space becomes an important element. The rest of the chapter is devoted to the study of an alternative which takes this into consideration, inspired by the concepts on Self-Adaptive Representations from EAs.

So, the underlying idea in the Self-Adaptive alternative proposed here is to select an initial search space and modify its size for each uncovered branch. More precisely, the region where the metaheuristic seeks for is initially defined with heuristic information obtained from the program's source code. During the process, the size of the region is

increasingly widened so that, if the optimum was not found in the current space, a new search is performed in a larger one. Next a detailed description of this approach is given.

The space of an objective branch is defined by the interval of values that each input parameter of a program can take. To be exact, for each branch and each parameter, a value is chosen to be the center of the interval, and a maximum increment over the center defines the amplitude. The process departs from a small range of values for each parameter and, as branches remain uncovered, the range is increasingly augmented. Centers of the intervals are fixed for the whole process, thus, in order to start seeking on a promising region, static heuristic information from the program is used to locate these points. In case this information is not useful to identify a center, a grid search method is applied.

Two approaches following this line have been developed. One of them adapts the size of the search space for all the uncovered branches at a time. In the other approach, each region enlargement involves a single objective branch.

(1) Assign initial search region to each branch
(2) **Repeat** until stopping criterion is met
(3) **Repeat** until stopping criterion is met
(4) $O \leftarrow$ Select objective branch
(5) Apply EDA to cover O
(6) Enlarge region

(1) Assign initial search region to each branch
(2) **Repeat** until stopping criterion is met
(3) $O \leftarrow$ Select objective branch
(4) **Repeat** until stopping criterion is met
(5) Apply EDA to cover O
(6) Enlarge region

Figure 6.2: Algorithms for the MOA (left) and SOA (right) approaches.

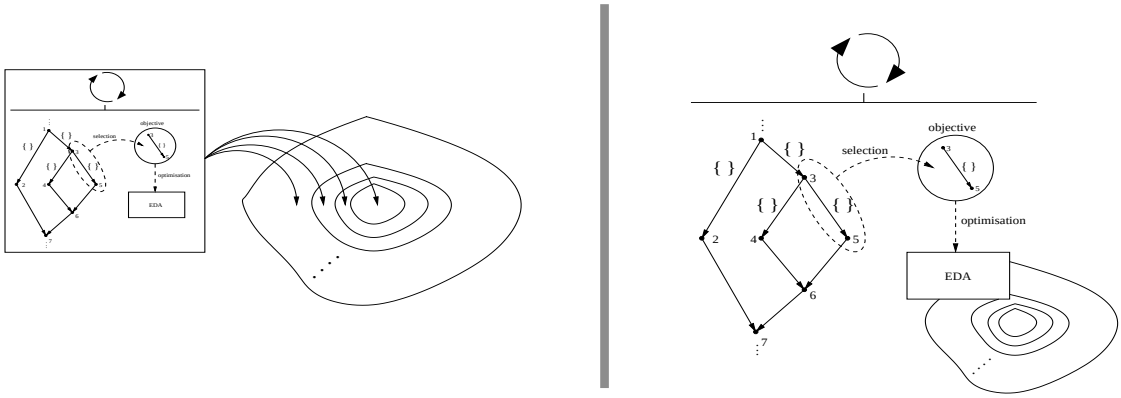


Figure 6.3: Schemas of the MOA (left) and SOA (right) approaches.

Multiple Objective Adaptation (MOA)

The idea behind this method can be clearly stated: to use the general scheme in Figure 3.5 over widening regions. This leads to the left side algorithm in Figure 6.2. Therefore, the basic approach is applied initially with a reduced interval of values for an input parameter and, once it is finished, if uncovered branches exist, it is applied again with a larger interval. The left side of Figure 6.3 depicts an illustration of this idea.

Single Objective Adaptation (SOA)

This alternative is similar to the basic approach except for the optimization step. Starting from a small search space, the EDA executes several times over increasingly augmented regions while the coverage of the objective branch is not attained. The right side of Figures 6.2 and 6.3 represent the algorithm associated with this method and a schema of the process, respectively.

In the next pages, these two approaches are discussed in detail by first explaining the steps of their algorithms, and later, how the set of inputs is managed.

6.3.1 Algorithm Steps Description

The description applies to both MOA and SOA, since the same steps for each algorithm implement the same concepts.

Region Initialization - step 1 (MOA, SOA)

Each branch is assigned an initial search region which will have the smallest size. A reduced region allows for a fast search, although the chances of containing the global optimum may be few. Hence, in order to reach a high degree of efficiency, it is important to obtain an initial region that is near the optimal input. Obviously, this is a difficult task, since the topology of the space should be known in advance (and no search would be required then).

Instead, it is possible to approximate the problem by using static heuristic information from the program's source code. Although different source code aspects could be regarded, in the present work, this information is obtained from the expression in the conditional statement corresponding to a branch. Assuming, with no loss of generality, that an input is composed of three parameters (a, b, c), then, the center of the initial region may be elicited through the following two heuristic rules:

- If an expression follows the form $F(a, b, c)$ **OP** K, where F is a known function of the input parameters, K is a constant and **OP** is a comparison operator, then the region is centered at point (C_a, C_b, C_c) such that $F(C_a, C_b, C_c) = K$.
- If an expression follows the form $F(a, b, c)$ **OP** $F'(a, b, c)$, where F and F' are known functions of the input parameters, and **OP** is a comparison operator, then the region is centered at point (C_a, C_b, C_c) such that $F(C_a, C_b, C_c) = F'(C_a, C_b, C_c)$.

Notice that the above rules refer to specific types of expressions. Many possibilities exist for the form of functions F and F' in an expression. For instance, it could depend on a number of source code variables or it might include calls to other programs. These rules constitute a first approximation to the problem by restricting F and F' to depend only on the input parameters, e.g. $F(a, b, c) = 7a + 25c$. Furthermore, each point (C_a, C_b, C_c) was calculated manually for the experimental programs employed to evaluate the present work. In order to reach complete automation of this step, a calculus tool could be employed, for example, Mathematica[1].

In case none of the above rules can be applied, the center of the initial region for a branch is obtained through a heuristic strategy based on the program's dynamic information; to be exact, a *grid search* method is employed. For each input parameter, the complete range of values is partitioned into τ intervals. The center of each of these intervals is taken as a reference value. Then, the inputs resulting from the combination of the reference values of all the parameters are evaluated with regard to the branch. The best input is selected as the center of the initial region. Notice that the granularity of the strategy may be tuned with the number of intervals τ, since the number of inputs generated is τ^p for a program with p parameters.

The idea behind a grid search is to explore a number of equally distant points from the whole search space. As τ grows, the number of points being considered approaches the complete number of points and, hence, the quality of the solution found might increase. However, reaching a certain value of τ may result in an unavoidable number of points. As a consequence, τ is regarded as a parameter of the approach. Figure 6.4 illustrates the strategy for the case of two parameters and $\tau = 6$; among the 6^2 points, the one inside the circle represents the input hypothetically chosen as the center.

Once the center is obtained using whichever of the strategies above, the specification of the initial search region of the branch is completed by defining an amplitude. This is achieved by setting, for each input parameter, an increment over the center. These initial increments are given as parameters to the test data generation system.

Thus, in essence, attending to the strategy used to elicite the initial region, we may classify branches in two types. On the one hand, those with the region centered at a

[1]Mathematica is a software package that solves equations symbolically. Web site: `http://www.wolfram.com/mathematica/`

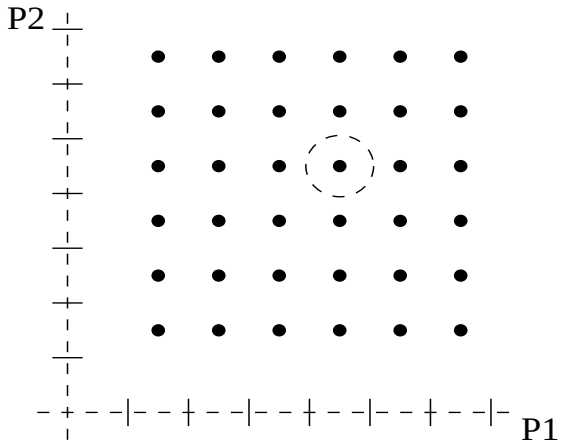


Figure 6.4: Schema of the grid search method.

point obtained through static heuristic information and, on the other hand, the branches with the region center chosen by means of a grid search, i.e. using dynamic information.

Stopping Criteria - steps 2 and 3 (MOA), steps 2 and 4 (SOA)

The stopping criterion at step 3, for MOA, and step 2, for SOA, refers to the general scheme (Figure 3.5). It is defined in the same way as in the basic approach, that is, full coverage achievement or unsuccessful treatment of every uncovered branch.

In contrast, the criterion in step 2, for MOA, and step 4, for SOA, alludes to the Self-Adaptive approach. Therefore, it states the point where the search space stops being enlarged. To obtain this point, a limit to the size of the region is given as a parameter to the system. Accordingly, in the case of MOA, the stopping criterion is to obtain full branch coverage or reach the size limit, while in SOA, the search stops when the objective branch is covered or the space attains its size limit.

Branch Selection - step 4 (MOA), step 3 (SOA)

The objective branch is selected following the strategy of the basic approach. Hence, the branch with the highest quality set of inputs at the moment is chosen, that is, the branch with the highest average objective function value over the inputs in the set.

EDA - step 5 (MOA, SOA)

The EDA seeks the optimal input in a search region centered at a fixed point. Therefore, an individual is a bit string representing an increment on the center of the current region. To be precise, the individual consists of a bit substring for each input parameter. Each substring represents an increment on the center of the interval of the corresponding parameter.

In the evaluation, the increment represented by the individual is added to the center of the region, resulting in the input for the objective function. In the current implementation of the approach, three parameter types are considered: integers, reals and characters. In the case of an integer, the bit substring represents the increment following a sign-magnitude codification. For real numbers, the IEEE floating point codification is used instead. In both cases, the input parameter value is obtained by summing the increment to the center. Finally, for a character type, a sign-magnitude codification is employed again in the substring. Then, the increment is summed to the center of the parameter, and the value obtained results in a character, according to the ASCII code table. Similarly, for more complex parameter types, an appropriate transformation could be defined to obtain the input parameter value.

As in the basic approach, the input is evaluated with regard to all the other uncovered branches and the sets of best inputs are updated accordingly.

The length of the individuals may vary between different EDA executions and, in consequence, it is not advisable to keep the same parameter values for the whole process. This is overcome by making some of the parameters adaptive [65].

A common practice in EAs is to fix the population size proportionally to the number of variables. For instance, in [153], several rules of thumb are suggested for a number of EDAs under specific conditions. In the present work, the population size is set at twice the length of the individual.

Additionally, it would be desirable to halt the search when no improvement can be obtained. This is a relatively unexplored matter in the field of EDAs, although a few recent works are emerging [160]. Here, a novel strategy has been developed. The problem is approximated by identifying the generation where the estimated probability distribution $p_l(\boldsymbol{x})$ is similar to the empirical distribution of the selected individuals. Thus, the criterion adopted is to stop the EDA when the Kullback-Leibler cross-entropy from $p_l(\boldsymbol{x})$ to $p(\boldsymbol{x})$ falls below a value α given as a parameter to the system.

Region Enlargement - step 6 (MOA, SOA)

The size of a search region is determined by the amplitude of the interval associated to each input parameter. In other words, this size is defined by a maximum increment

on the center of the interval of each parameter. In the EDA, an increment for each parameter is represented as a substring of bits. Therefore, the number of bits in each substring specifies the size of the region.

The search region is enlarged by augmenting the amplitude of the interval associated with a chosen input parameter. A bit is added to the substring representing the next parameter in the order given by the input, from left to right.

6.3.2 Management of the Set of Inputs

The control of the set of inputs of each branch introduces disparities between the approaches which require a separate explanation.

Operation in MOA

In the MOA alternative, during the EDA execution, it is possible that an input being evaluated for a branch distinct from the objective falls outside the current search space. Therefore, when the branch is selected as the new objective and the EDA is to be initialized with the inputs in the set of the branch, some of these inputs might be out of the region.

Hence, instead of using only one set of inputs, two sets are associated with each branch. One of them keeps the best inputs inside the current search region - *inside set* - and the other one, those falling outside - *outside set*. This implies that, during the evaluation in the EDA, the input is stored in the required set and, this way, the initialization is directly performed from the inside set. More precisely, for each input in the set, the corresponding increment on the center is obtained (in its binary form) and added to the population.

In order to maintain the sets, before starting a new run of the general scheme (step 3), the inside set is updated with the inputs in the outside set which belong to the new region.

Operation in SOA

Regarding the SOA approach, each time the EDA is executed, the search region is different from the previous. In this situation no advantage is obtained with two sets, so just one containing all the inputs is used.

To initialize the EDA, firstly, the increments associated with the inputs in the set are calculated. Then, the increments inside the current region are included in the population. Those falling outside are truncated to fit into the region and, then, are added to the

population. A possible disadvantage of this strategy is that, as the population converges to similar individuals, if these are high quality solutions, they will be included in the set. Thus, initialization for the next region might cause a low diversity between individuals and result in a poor search. With the intention of alleviating this phenomenon, half of the EDA's initial population is randomly generated.

Another problem in SOA concerns the retrieval of the initial search region for the EDA. If the objective branch is selected for the first time, the initial region is given by its center and the initial increment. However, it can so happen that, in the EDA evaluation, the input enters the set of a branch already treated and, therefore, makes this branch a candidate objective once again. Supposing that the branch is selected for a second time, the initial search region should not be taken as before, since the new inputs in the set could be in a larger space and, hence, would not be used to seed the population. The solution adopted here has been to obtain the initial region size of the smallest new input in the set.

6.4 An Execution Example

As an illustration of the approach, some steps of an hypothetical execution of MOA and SOA are explained next. The example of Figure 3.6 will be used once again. Hence, test cases are to be generated for a program where an input is composed of three integers a, b and c.

First of all, both algorithms require the assignment of an initial search region to each branch (step 1). Thus, for each branch and input parameter, an initial interval of values must be defined. This is attained by fixing the center of the interval and an increment on the center.

Two strategies are proposed for the center elicitation: static information and dynamic information based. The branch represented by arc $(2,3)$ in the graph is associated to condition `if((b*b)-(4*a*c)<0)`, so the static strategy is used. A point satisfying $b^2 - 4ac = 0$ is chosen as the center, for instance, $(0,0,0)$. In contrast, the condition of branch $(7,8)$ is `if((a*x*x+b*x+c)==0)`, so the grid search method must be employed. If an integer is codified with 15 bits in two's complement representation, the complete interval of values of each parameter is $[-32768, 32767]$. With $\tau = 8$, the number of points generated and evaluated is $8^3 = 512$. The best is $(4095, 4095, -20480)$, which is taken as the center of the region.

Once the center of each branch is fixed for a, b and c, the initial region is obtained with an increment on each center. To keep the example simple, 5 bits are given to represent an increment for each input parameter, resulting in a maximum increment of ± 31. Thus, the initial region for branch $(2,3)$ is $[-31, 31] \times [-31, 31] \times [-31, 31]$ and for branch $(7,8)$ it is $[4064, 4126] \times [4064, 4126] \times [-20511, -20449]$.

MOA Example

MOA applies the basic approach (steps 3 to 5) over increasing search regions until a maximum size is achieved (step 2). Using a maximum of 10 bits to represent an increment for each input parameter, the maximum region for branch $(2,3)$ is $[-1023,1023] \times [-1023,1023] \times [-1023,1023]$ and for branch $(7,8)$ it is $[3072,5118] \times [3072,5118] \times [-21503,-19457]$.

Now, assume that the size of the region in the current round is defined with 7 bits for a and b, and 6 bits for c. This implies that, in the previous region, 6 bits were used for b.

Remember that two sets of inputs are associated to each branch: the inside set and the outside set. The selection strategy (step 4) chooses the branch with the highest quality inside set. If branch $(2,3)$ was selected, the initial population of the EDA (step 5) is seeded with the inside set of this branch. In this particular case, an individual representing the increment $(98,-34,15)$ would result in input $(98,-34,15)$, as the region center is $(0,0,0)$. Aside from calculating the objective function value of this input, it is also evaluated for the rest of the branches. For instance, evaluating the input for branch $(7,8)$ implies that its associated increment must be induced. Thus, input $(98,-34,15)$ results in increment $(-3997,-4129,20495)$ for branch $(7,8)$. To represent such an increment, 12 bits would be necessary for parameters a and b, and 15 bits for c, so it falls outside the current region. In consequence, the outside set is updated if its quality is improved with this input.

Once the basic approach finishes without covering all the branches, the current region is enlarged. In the previous region the interval of b was increased, so now c is augmented to 7 bits, resulting in a search region where a, b and c represent an increment with 7 bits.

SOA Example

In SOA, the optimization phase is applied over increasing regions (steps 4 to 6). The rest of the steps are those in the basic approach, so they are not illustrated here. From now on, the following is assumed. Branch $(2,3)$ is selected as the objective and the current region of the optimization phase is defined with 7 bits for a and b, and 6 bits for c.

In this algorithm, only one set of inputs is maintained for each branch during the process. Half of the EDA's initial population is randomly created and the other half is seeded from the inputs in the set. For instance, in order to seed the population with input $(509,-11,35)$, the increment associated to branch $(2,3)$ must be induced first. The result is increment $(509,-11,35)$ (remember that the center was $(0,0,0)$). This increment falls outside the current region because 9 bits are needed to codify the 509. Therefore, the bit substring representing the 509 is truncated to 7 bits to fit in a's interval. In contrast, an input $(45,117,-21)$ would result in the increment $(45,117,-21)$, which is inside the current region and is to enter directly in the initial population.

As in the basic approach, once the value of an input is obtained for branch $(2, 3)$, it is evaluated for the remaining branches. If the quality of the set of the branch is improved, then the input enters the set.

After the EDA finishes, the current region is enlarged in the way described above for MOA.

6.5 Performance Evaluation

In order to observe the performance of the presented approaches, test cases were generated for all the programs taken for experimentation in the basic approach. The goal of the evaluation was threefold: analyzing the behavior of the approaches, comparing their results with those attained by other alternatives, and checking whether they constitute a solid alternative in the real world. Regarding the former goal, performance of each algorithm, MOA and SOA, was studied in isolation. In the second goal, three topics were considered. Firstly, MOA was compared to SOA. Then, the static information based heuristic employed to define the initial search region was compared to the dynamic one. Finally, MOA and SOA results were faced with those by the basic test data generator. For the later goal, MOA and SOA were evaluated over a number of "real-world" programs.

6.5.1 Experimental Setting

Recall that, among the EDAs evaluated in Chapter 4, TREE was concluded to show the best performance overall. In consequence, TREE was the EDA chosen here for the optimization step in both MOA and SOA approaches. At each generation, half of the population was selected according to a rank-based strategy. New individuals were simulated by means of Probabilistic Logic Sampling, and the population was created in an elitist way. The objective function employed in the experiments was formulated according to equation 3.2. Notice that the stopping criterion adopted for the EDA seems to be specially suitable for TREE. This algorithm obtains the tree dependent factorization minimizing the Kullback-Leibler divergence to the empirical distribution. Since the EDA stops when this divergence value is lower than α, the value of the optimal model is directly being considered. For the experiments, α was determined after a number of preliminary executions.

Other system parameters that need to be fixed are the size of the initial and the largest possible region. Given a program, this is achieved by setting, for each input parameter, the minimum and maximum possible amplitude of its associated interval of values. Obviously, a different amplitude may be linked to each input parameter and, thus, the shape of search regions could be controled. However, for the experiments, no a priori

knowledge is assumed and, therefore, amplitude values were kept constant for all the input parameters of a program.

Table 6.2 presents, for each program, the values selected for the system parameters, i.e. number of bits for the increment on the initial region (minimum), number of bits for the increment on the largest allowed region (maximum), and α value.

Also shown in Table 6.2 is the number of branches, and for how many of them the centers of each input parameter were obtained through the static information based (static) and the dynamic information based heuristic (dynamic). As it can be seen, in all the programs but three, almost every branch is static. `Remainder` is relatively balanced in this sense, while in `Atof` outstanding branches are dynamic.

program	characteristics			parameters		
	branches	static	dynamic	minimum	maximum	α
`Triangle1`	26	24	2	5	15	2
`Triangle2`	26	24	2	5	7	2
`Triangle3`	20	16	4	5	10	2
`Triangle4`	26	20	6	5	10	2
`Atof`	30	2	28	5	7	25
`Remainder`	18	10	8	5	16	5
`Complexbranch`	22	18	4	5	10	15

Table 6.2: Experimental programs characteristics and parameters in the experiments.

Remember that the dynamic information based strategy consisted of a grid search. In this method, the value of parameter τ defines the number of inputs being considered candidate centers. More precisely, for a program with p parameters, τ^p inputs are created for evaluation. Alternatively, the larger the τ, the finer the granularity of the strategy and, hence, the chances of finding a high quality initial search region increase. In the experiments, τ was set from 1 up to 5 for all the programs excepting `Atof`, which used τ up to 3.

Additionally, in order to avoid too long executions, a limit of 150000 inputs generated was established. As soon as this limit was detected, the experiment was forced to terminate.

6.5.2 MOA Performance

Table 6.3 presents the results of the experiments conducted. For each value of τ and each program, the table collects the average values in ten executions for the percentage of covered branches (%) and the number of inputs generated during the process (#). The best values of τ for each of these two measures and each program are highlighted in gray.

τ	`Triangle1`		`Triangle2`		`Triangle3`		`Triangle4`		`Atof`		`Remainder`		`Complexbranch`	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
1	100	212	99.62	579	100	302	100	2223	98.33	68936	99.44	629	95†	149952
2	100	282	99.62	995	100	338	100	1967	43.33†	150062	98.89	1628	100	1856
3	100	190	99.23	1143	100	311	100	2436	96.33	134849‡	100	186	95.45†	102711
4	100	440‡	99.23	880	100	285	100	1922	-	-	100	84	100	16096‡
5	100	381‡	100	990	100	330	100	1738	-	-	100	57	95.91†	117213

Table 6.3: Results of the MOA approach.

As can be noticed, in all programs except `Atof`, full coverage is reached. `Atof` seems to be the hardest, since the lowest coverage and the largest number of inputs are attained in this program.

Overall Performance Analysis

Regarding Table 6.3, no apparent relation exists between τ and the best results, since these are obtained with alternative values of τ, ranging from the lowest to the largest value.

In order to validate the best performance values in MOA, an analysis based on statistical tests was conducted. Since coverage is a primary measurement, for each program and each value of τ, the Mann-Whitney test with regard to the best τ value (in gray) was applied to the coverage results. Then, for the cases where no difference was found, the test was again used over the number of inputs generated. Table 6.3 presents the outcomes of these tests; symbol '†' denotes the cases where coverage dissimilarities ($p < 0.01$) were found, while '‡' refers to the number of inputs.

In less than half the cases (8 from up to 26 possibilities), the best values of τ constitute an improvement with statistical evidence. It can be seen that statistically significant differences were obtained for the coverage reached in `Atof` and `Complexbranch` for a few values of τ. In the number of inputs generated, a few dissimilarities were observed in `Triangle1`, `Atof`, and `Complexbranch`. Hence, considering these results, we cannot conclude whether the best τ makes a difference.

Initial Region Heuristics Performance

According to the previous analysis, no clear conclusion can be stated on the most suitable τ value for a program. In order to better understand the relevance of τ in the results, it could be interesting to examine the influence of the initial heuristics used to elicit the initial regions.

Table 6.4 shows, for each program, the number of branches covered ($\#_o$) and number of inputs generated ($\#_i$) by the static and dynamic heuristics from the region initialization

τ	`Triangle1`		`Triangle2`		`Triangle3`		`Triangle4`		`Atof`		`Remainder`		`Complexbranch`	
	$\#_o$	$\#_i$	$\#_o$	$\#_i$	$\#_o$	$\#_i$	$\#_o$	$\#_i$	$\#_o$	$\#_i$	$\#_o$	$\#_i$	$\#_o$	$\#_i$
static	2	12	2	12	1	8	1	10	1	1	1	5	10	9
dynamic, $\tau = 1$	0	1	0	1	0	1	0	1	0	1	0	1	0	1
dynamic, $\tau = 2$	1	8	0	8	2	8	3	8	0	1024	7	4	2	64
dynamic, $\tau = 3$	0	27	0	27	2	27	3	27	0	59049	7	9	2	729
dynamic, $\tau = 4$	1	64	0	64	3	64	6	59	-	-	8	14	2	4096
dynamic, $\tau = 5$	1	125	0	125	3	125	6	119	-	-	8	22	2	15625

Table 6.4: Results of the initial region obtainment heuristics.

phase. The first row presents the values of the static heuristic, while the rest correspond to the dynamic heuristic (grid search) with the different values of τ. Notice that the overall contribution of these heuristics consists of the sum of the static and the dynamic results for a chosen τ. For instance, in `Triangle1` with $\tau = 2$, after applying the static and dynamic strategies, 2+1=3 branches were covered (which implies a 11.54% coverage) and 12+8=20 inputs were generated.

It can be seen that, regarding the static strategy, a number of branches are covered in all the programs just by the application of the two heuristic rules. Moreover, in some cases this is a significant number. In `Complexbranch`, 10 out of the 18 static branches are covered, and in `Atof`, one of the two static branches are attained. Anyhow, considering that most of the branches are static in the main body of the programs and that 100% coverage was obtained in almost all of them, the heuristic rules appear to be effective.

The dynamic heuristic is a grid search method. In such a method, given a problem, as τ increases, more points are generated and the quality of the best solution found is expected to grow. In the context of the test data generator, this implies that the number of branches covered is expected to increase with growing values of τ. However, a main drawback of a grid search is that the value of τ needed to reach an outstanding solution may be large, producing a prohibitive number of solutions. This could be the case even for small values of τ if the number of problem variables is relatively big [7]. Accordingly, Table 6.4 shows alternating behaviors. In `Triangle3`, `Triangle4` and `Remainder`, the coverage increases as τ grows, while, for the rest of the programs, this is not held. Moreover, comparing the values in Table 6.3 and Table 6.4 for `Triangle1` and `Triangle3`, it can be observed that, with $\tau = 5$, a significant part of the inputs is generated by the grid method; the same occurs in `Atof` with $\tau = 3$. In consequence, results do not necessarily improve by increasing the value of τ.

This observation can also be extrapolated to the best overall results in Table 6.3, since these are obtained with different values of τ. Furthermore, recall that, in the previous statistical analysis, no significant influence of τ on the best coverage values was found, excepting a few cases. Thus, these results suggest that the effect of the grid search is neutralized by the rest of the phases in MOA.

τ	`Triangle1`(26)	`Triangle2`(26)	`Triangle3`(20)	`Triangle4`(26)	`Atof`(30)	`Remainder`(18)	`Complexbranch`(22)
1	2.2	2.3	2.7	5.1	15.2	0.7	2.5
2	2	3.4	1.9	5.3	17	0.8	1.9
3	1.7	3	2.1	4.9	15.5	0.5	1.5
4	2.4	2.5	2.4	4	-	0.6	2.1
5	1.9	2.8	2.1	4.7	-	0.4	1.6

Table 6.5: Average number of branches sought in the MOA approach.

Region Enlargment Performance

Another factor that contributes to the performance of the generator is the number of region enlargements. New regions may include uncovered objectives. Instead, as more increments are carried out, the number of inputs created is expected to grow, since more search steps are executed.

During the experiments, each run was recorded with the purpose of studying how the system operates. Using this information, given a search region and an objective branch, the number of inputs generated and whether the objective was covered or not can be elicited. This is shown in Figure 6.5. The graphics above relate to the number of branches covered in each region. More specifically, they only consider the branches which were covered by the initial region heuristics or those selected as objectives and covered by the EDA. Notice that not all the branches need to be explicitly searched, because during the fitness evaluation in the EDA, branches distinct from the objective may eventually be covered. Thus, in each graphic ahead of Figure 6.5, the x-axis takes values in the range of possible regions, while the y-axis concerns the number of branches covered by the initial heuristics or by the EDA. The points depicted correspond to the results (averaged over the ten executions) in each region, for each value of τ. To finish with the specification, Table 6.5 shows the average total number of branches searched by the EDA; aside from a program name, the number of branches in the program is provided in brackets. Analogously, the bottom part of Figure 6.5 presents the accumulated average number of inputs generated (y-axis) for each region (x-axis), given a value of τ and a program.

As can be observed in the upper half, with the exception of `Atof`, almost all the branches were attained in the very first search regions. To some extent, this is not surprising, since the first region includes the coverage of the initial heuristics and the EDA, while the rest of regions only involve the EDA contribution. In the bulk of the programs, the number of static objectives is high (see Table 6.2), so the graphics suggest that the static information based heuristic used to elicit the initial region is an adequate strategy. Indeed, this could be the cause of the poor behavior of `Atof`, since it contains a reduced number of static branches. Moreover, owing to the quite large set of parameters of an input in this program, τ only takes values up to 3, which seems to be insufficient for the grid search to obtain a promising initial center. Another program with a relevant

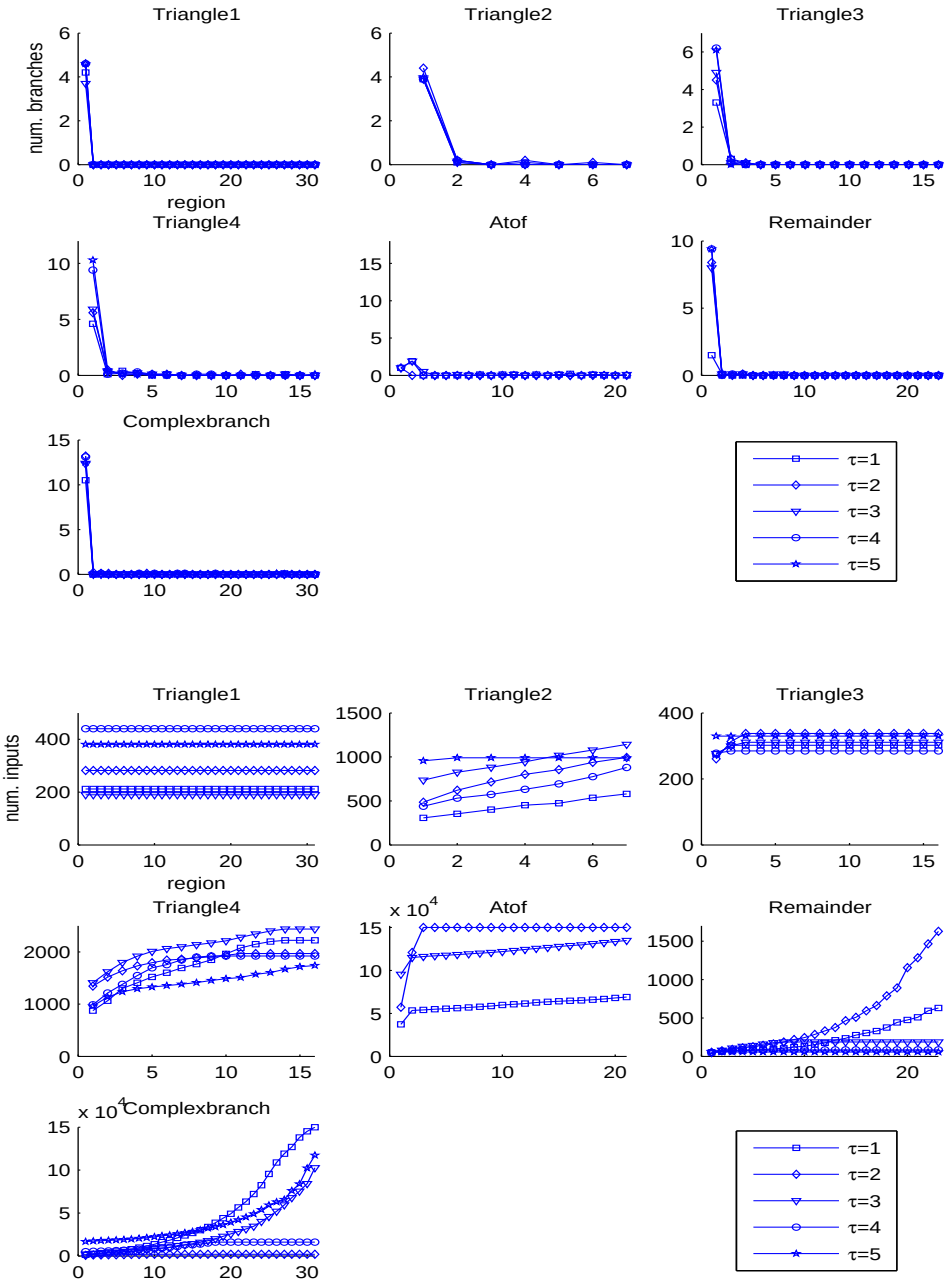


Figure 6.5: Average number of branches covered (above) and inputs generated (below) for each region in MOA.

number of dynamic branches is `Remainder`. In this case, both the dynamic and the static heuristics appear to behave successfully, as all the dynamic branches were covered directly by the grid method (see also comments on Table 6.4) and most of the static ones were attained in the initial region.

Anyhow, the effect of the different search spaces should not be underestimated. In 5 of the 7 programs, a few objectives are still covered in advanced regions and, therefore, the coverage measurement grows.

As for the inputs generated, Figure 6.5 below shows that their number stays relatively low at the initial stages, although it increases as branches remain uncovered. If complete coverage is attained, the curve stabilizes, otherwise, it keeps growing. More specifically, the curve grows smoothly in a number of cases (e.g. `Triangle2`), although for other instances it augments rapidly with certain values of τ (`Complexbranch` and $\tau = 1$, for example). In these last cases, the latter regions offer more promising solutions than in the previous stages and the search intensifies. This means that the EDA operates for a larger number of generations and, thus, more solutions are generated. This can be clearly remarked in the `Remainder` and `Complexbranch` programs. The low coverage reached by `Atof` for $\tau = 2$ can be understood by observing the number of inputs generated. The figure reveals that the limit of 150000 inputs was attained in the early regions, so the generator stopped prematurely and no more objectives could be covered (see `Atof` above).

To summarize, it could be deduced that, on one side, the search over different regions allows the MOA generator to obtain the highest coverage (effectiveness). On the other side, the answer of the dynamic heuristic seems to be more unstable than for the static information based strategy. In fact, the high quality values of the early spaces suggest that the static heuristic is useful to achieve objectives soon and, therefore, generate a reduced number of inputs (increase efficiency). In order to shed more light on this matter, this will be further studied in a following analysis in Section 6.5.4.

6.5.3 SOA Performance

Apropos the SOA algorithm, Table 6.6 shows the results of the experiments for the programs. The cell format is the same as in Table 6.3. Similar to the MOA approach, the most difficult program for the test case generator is `Atof`. However, in this case, 100% coverage could not be obtained for `Triangle4`.

Overall Performance Analysis

Statistical tests were used to identify the best performance values. Thus, the null hypothesis of equal distribution densities between the best τ values and the others was evaluated in the manner explained in the previous section.

τ	Triangle1		Triangle2		Triangle3		Triangle4		Atof		Remainder		Complexbranch	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
1	100	401	100	333	100	250	99.23	3630	96	65078	100	58‡	95.91†	122813
2	100	282	100	374	100	237	98.46	5476	49.33†	150049	100	33	100	18406
3	100	246	100	293	100	222	99.23	6003‡	95	133649‡	100	30	98.18	107144‡
4	100	391	100	266	100	297	98.46	2663	-	-	100	59‡	98.18	68485
5	100	399‡	100	765‡	100	314‡	99.23	2546	-	-	100	91‡	97.73	106698‡

Table 6.6: Results of the SOA approach.

Differences were statistically significant ($p < 0.01$) with regard to the coverage measurement in a pair of cases (`Atof` with $\tau = 2$ and `Complexbranch` with $\tau = 1$). In contrast, for the number of inputs generated, the 10 differences obtained ($p < 0.01$), from up to 24 possibilities, spread over all the programs. The outcomes from this analysis reinforce the conclusions of the MOA approach. Taking the programs used here into account and respecting the best results, the τ value has no significant influence regarding the coverage measure. On the other hand, for the number inputs, not enough dissimilarities to make a reliable conclusion were found.

Initial Region Heuristics Performance

MOA and SOA share the same initial region elicitation step. Therefore, results in Table 6.4 also apply here, as well as the comments on the behavior of the static and dynamic heuristics.

Concerning the lack of influence of τ on the best coverage results, the outcomes of SOA are almost equal to those in MOA. In consequence, here, the corresponding reason is suggested, that is, the remaining steps of SOA cancel the effect of the grid search.

Region Enlargement Performance

Accordingly to the MOA alternative, the experiment executions were monitored and the values raised by a search step were stored. Figure 6.6 reveals, for each possible search region, the average number of objectives covered by the initial heuristics and the EDA (above), and the inputs generated (below) during the process. The figure format is the same as in the previous section. Table 6.7 assists in the understanding of the figure by presenting the average total number of branches searched by the EDA and, in brackets, the number of branches in a program.

Drawing a rough comparison of Figure 6.6 and Figure 6.5, it can be noticed that, in general, the behavior of both approaches is similar. Although differences appear with some programs (`Remainder` in the number of inputs), the remarks on the MOA algorithm can also be applied to SOA.

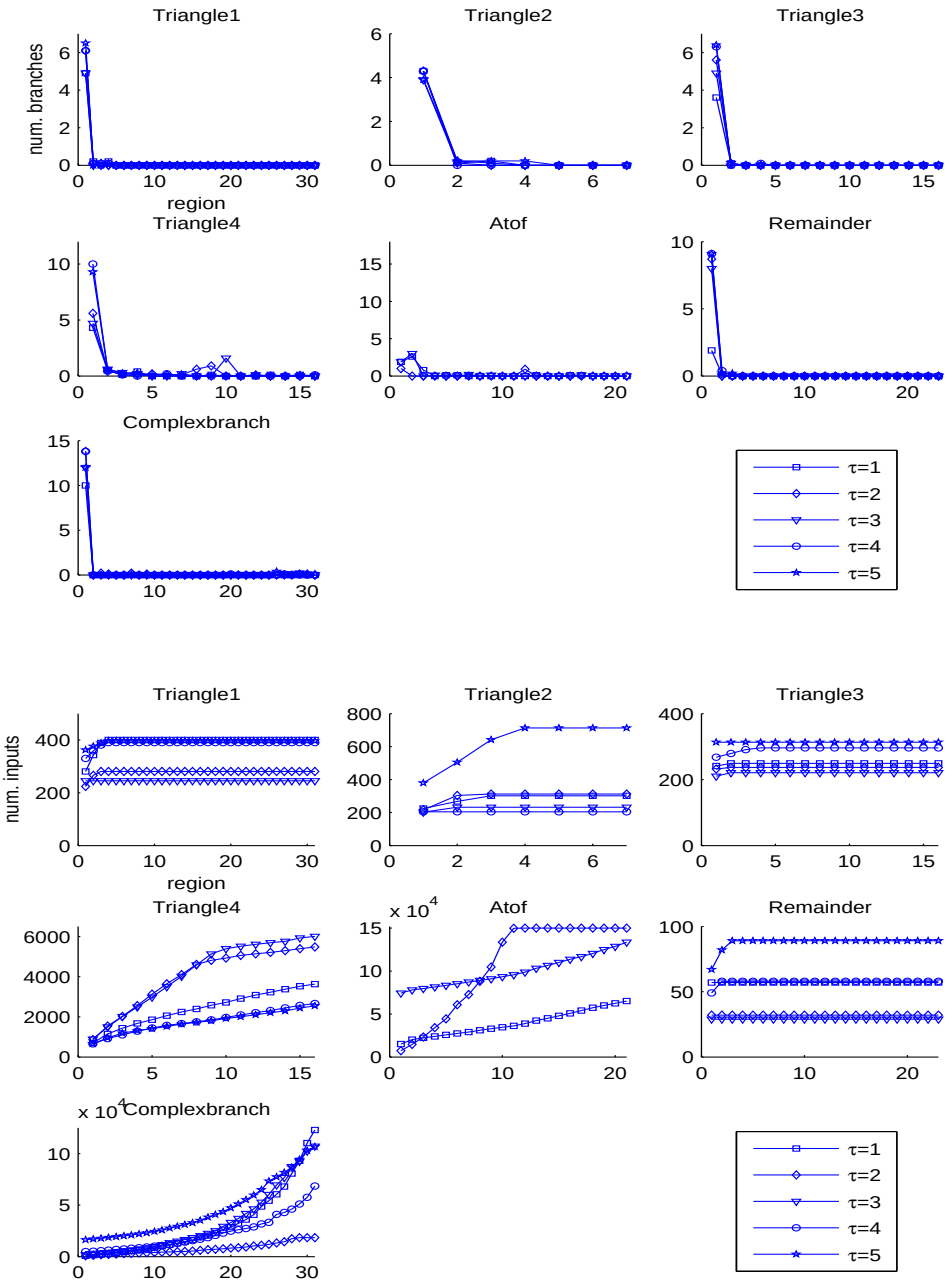


Figure 6.6: Average number of branches covered (above) and inputs generated (below) for each region in SOA.

τ	Triangle1(26)	Triangle2(26)	Triangle3(20)	Triangle4(26)	Atof(30)	Remainder(18)	Complexbranch(22)
1	3.4	2.6	2.7	5.1	6.7	1	1
2	3.3	2.6	2.7	4.9	1.9	0.7	2.6
3	2.9	2	2	4	6.6	0	1
4	3.3	2.3	2.4	4.4	-	0.5	2.5
5	3.6	2.5	2.4	3.8	-	0.5	1

Table 6.7: Average number of branches sought in the SOA approach.

6.5.4 MOA vs. SOA vs. Other Approaches

Next, each Self-Adaptive algorithm is compared to other approaches to evaluate its performance and know if it represents a competitive alternative.

MOA vs. SOA

In the MOA approach, each region enlargement concentrates on the test case generator as a whole. In contrast, each increment of the SOA alternative refers to an independent EDA search phase. Therefore, a formal comparison of both algorithms in terms distinct from the coverage and inputs generated becomes a difficult task. However, it might be suspected from the common conclusions raised in Sections 6.5.2 and 6.5.3, and from the matching behavior shown in Figures 6.5 and 6.6, that important similarities exist between them.

In order to know whether MOA and SOA offer a similar behavior in terms of coverage and inputs created, Table 6.3 and Table 6.6 were used to find statistically significant differences between the results. To be precise, the Mann-Whitney non-parametric test was applied to each approach and value of τ. Considering coverage, the null hypothesis of equal distributions was rejected ($p < 0.01$) only for `Atof` with $\tau = 2$ and `Complexbranch` with $\tau = 3$. For the number of inputs generated, differences ($p < 0.01$) were obtained in six cases: `Triangle1` with $\tau = 1$, `Triangle2` with $\tau = 3$, `Triangle3` with $\tau = 3$, `Triangle4` with $\tau = 2$ and $\tau = 3$, and `Remainder` with $\tau = 3$. Since half of the best result values in these cases corresponded to each approach, it cannot be stated which one behaves better.

According to the tests, it may be concluded that, excepting a few cases, the performance of MOA and SOA algorithms is similar in terms of coverage and number of inputs generated.

Static vs. Dynamic Information Centers

An element which appears to be important in the Self-Adaptive approach is the initial search space. If this is located in an adequate region, the effort in finding the optimal

τ	Triangle1		Triangle2		Triangle3		Triangle4		Atof		Remainder		Complexbranch	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
1	96.15†	150032‡	100	1688‡	100	453	100	1497	99	89230	99.44	838	95.45	145272
2	76.92†	150024‡	11.54†	150024‡	100	5324‡	94.62†	64131‡	43.33	150050	99.44	7825‡	90.91†	150047‡
3	11.54†	150020‡	11.54†	150017‡	99	7829‡	88.85†	44311‡	92.67	138653	100	49	95.45	112269
4	100	61916‡	11.54†	150018‡	100	154‡	93.85†	29048‡	-	-	100	6726‡	97.73	114918‡
5	100	74577‡	11.54†	150024‡	100	185‡	91.54†	37602‡	-	-	100	42‡	95.45	134630

Table 6.8: Results of the MOA approach with no static information based initial centers.

τ	Triangle1		Triangle2		Triangle3		Triangle4		Atof		Remainder		Complexbranch	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
1	94.62†	150026‡	100	2314‡	100	376‡	98.46	3778	95	91148	100	71	98.18	87891
2	76.92†	150023‡	23.08†	150024‡	100	3691‡	89.62†	28819‡	47.33	150064	100	9931‡	93.64†	128868‡
3	56.15†	150028‡	34.62†	150020‡	99	4442‡	89.23†	28566‡	95.67	125487	100	33	97.27	111895
4	100	25031‡	65†	145067‡	100	125‡	89.23†	24322‡	-	-	100	8377‡	96.36	113230
5	100	30808‡	23.08†	150014‡	100	185‡	90†	25134‡	-	-	100	42‡	98.18	116060

Table 6.9: Results of the SOA approach with no static information based initial centers.

solution may be low. However, if the EDA departs from an unsuitable region, a huge number of interval increments could be necessary to reach the optimum, or it could not even be attained. In the present work, the definition of the initial space of each branch is based on static or dynamic heuristic information. In order to compare these two strategies, the previous experiments were repeated changing static information based centers to be dynamic information based. Tables 6.8 and 6.9 show the results for the MOA and SOA algorithms, respectively.

The differences between the static and dynamic strategies for the coverage and number of inputs were studied through statistical tests. In other words, the Mann-Whitney test was employed to evaluate the equality between the distribution densities of the algorithms with and without static strategy. Similar to previous tables, the symbols '†' and '‡' beside a cell in Table 6.8 denote a statistically significant difference ($p < 0.01$) between the experiments in the cell and the corresponding values in Table 6.3. Analogously, the same applies to Table 6.9 and Table 6.6.

As can be observed, in MOA, the differences associated with the coverage concentrate on three programs: `Triangle1`, `Triangle2` and `Triangle4`. However, concerning the number of inputs generated, from up to 33 tests, dissimilarities were obtained in 23 cases. All in all, the programs with a large proportion of static to dynamic branches (see Table 6.2) offered differences, excepting `Complexbranch` for a few values of τ which shown an inferior performance in Table 6.3. In contrast, the programs with a more significant number of dynamic branches, revealed, in general, fewer dissimilarities. In the SOA algorithm, differences were found in almost the same cases as in MOA.

Regarding these significantly different instances in Tables 6.3 to 6.9, it can be noticed

that in almost all of them, the best results correspond to the approach using the static strategy. The only exceptions are `Triangle3` with $\tau = 4$ and $\tau = 5$, and `Remainder` with $\tau = 5$ for both MOA and SOA, in the number of inputs generated.

The remarks from these tests are captured by Figures 6.7 and 6.8 for MOA and SOA, respectively. In each graphic, the different objectives are represented in the x-axis, while the y-axis takes values in the range of possible region enlargements. Thus, given a program, the graphic in the upper half in a figure shows the number of increments performed for each static (labeled with a cross) and dynamic (labeled with a circle) objective. To be exact, the average and the standard deviation over τ and the ten executions are depicted for each objective. Analogously, in the bottom part of a figure, the values associated with the variant using only dynamic objectives are presented.

Both figures show clear disparities between the *static-dynamic* and the *dynamic* approaches in programs where the bulk of the statistical tests observed differences (that is, `Triangle1`, `Triangle2`, `Triangle4`). In contrast, in `Atof`, where no significant dissimilarity was found, behavior is almost the same. Remaining programs fall somewhere in between; they respond differently for a few objectives, although, in most of them, response is alike.

The significant differences obtained in the number of inputs generated are also reflected by the figures. In all the statistically distinct programs, the sum of the average number of increments in the dynamic approach is larger than in the static-dynamic one. Indeed, it can be noticed that the main body of the objectives where changes occur between both approaches corresponds to static cases which had turned out to be dynamic.

Thus, it may be concluded that the suggestions raised in Section 6.5.2 on the static information based strategy are confirmed. This strategy can make a difference in the coverage reached but, most of all, in the number of region enlargements and, consequently, in the number of inputs created. Moreover, the static heuristic improves or equals the dynamic one, with the exception of a few cases.

Self-Adaptive vs. Basic Approach

In order to have an idea of the quality of the results of the Self-Adaptive alternative, they were compared with those obtained by the basic test data generator.

The range of input parameter values for the basic approach was obtained centering the interval in 0 and adding the maximum increment shown in Table 6.2. To make the comparison as fair as possible, the EDA chosen was TREE and its parameters were the same as in Section 6.5.1, apart from two of them. The population size and the maximum number of generations were fixed with the values in Chapter 4 offering the best performance for TREE.

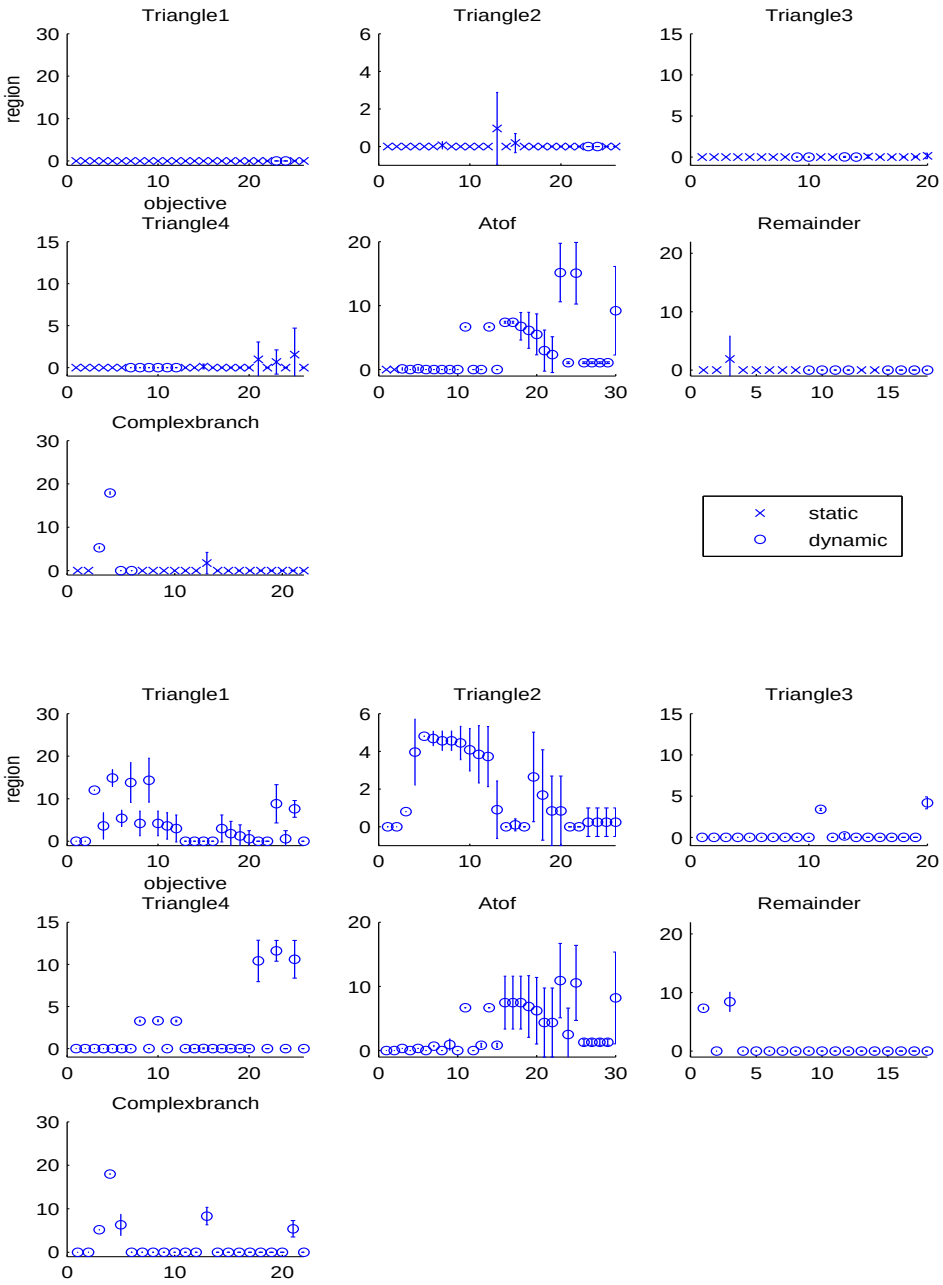


Figure 6.7: Average number of region enlargements per objective in MOA (above) and MOA with no static objective (below).

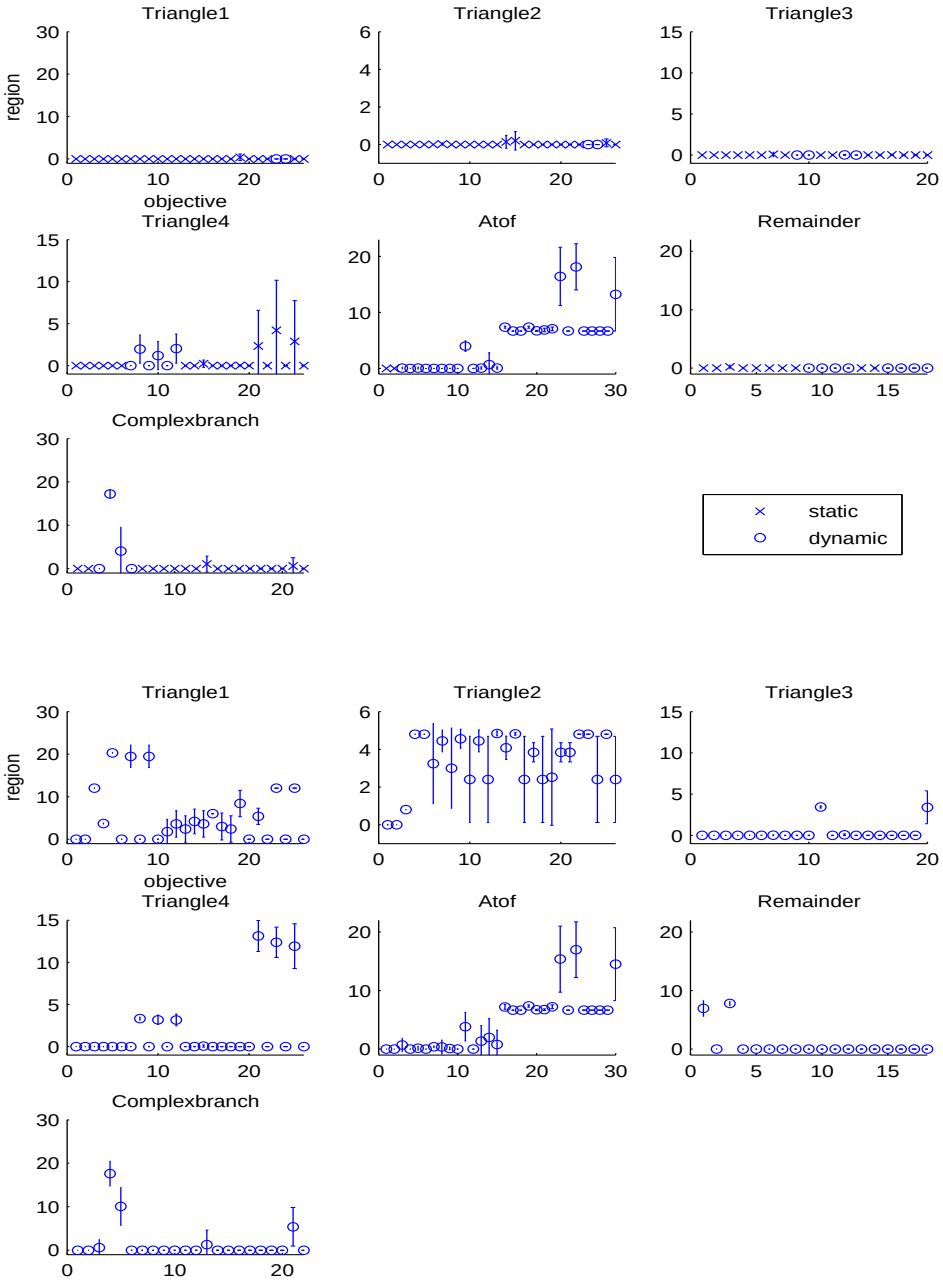


Figure 6.8: Average number of region enlargements per objective in SOA (above) and SOA with no static objective (below).

τ	`Triangle1`		`Triangle2`		`Triangle3`		`Triangle4`		`Atof`		`Remainder`		`Complexbranch`	
	%	#	%	#	%	#	%	#	%	#	%	#	%	#
Basic	99.62	9940‡	100	5880‡	100	4180‡	99.62	51560‡	100	3475	100	1240‡	98.64	21420‡
MOA	100	190	100	990‡	100	285	100	1738	98.33	68936‡	100	57‡	100	1856
SOA	100	246	100	266	100	222	99.23	2546	96†	65078	100	30	100	18406

Table 6.10: Best results of the basic, MOA and SOA approaches.

Table 6.10 shows the best values (with priority to coverage) of the MOA, SOA and basic approaches. The outstanding results are highlighted in gray.

It can be observed that the Self-Adaptive alternative outperforms the basic approach in the coverage reached as well as the number of generated inputs in all the programs except `Atof`. In fact, the poor behavior shown in the results of previous tables for this program becomes evident here, mostly with regard to the number of inputs. In `Atof`, a number of objectives can only be covered when the largest search region is reached. Since the Self-Adaptive approach departs from a reduced region and the grid search method seems to provide an unsuitable initial center, performance is worse than for the basic alternative, which operates over the largest region directly.

The purpose of the current comparison is to identify the approach offering the best performance. Hence, the statistical analysis explained in Section 6.5.2 was used to validate these results. Similarly to the previous table, Table 6.10 provides the outcomes of the analysis.

Significant differences ($p < 0.01$) in the coverage values were noticed just for `Atof`, between the basic approach and SOA. In contrast, MOA revealed a difference in this program for the number of inputs generated. Thus, it can be deduced that the basic generator improves SOA and MOA with statistical evidence in `Atof`. In spite of this, for the rest of the programs, the basic approach presents dissimilarities ($p < 0.01$) in the number of inputs created with regard to the best result.

Therefore, it can be inferred that in almost all the programs the Self-Adaptive approach outperforms the basic one.

6.5.5 Evaluation with Real-World Programs

The experiments conducted in the previous sections involve typical programs which are known to include several challenging branches. Obviously, test data generation for "real-world" programs may be as difficult, although it could result in a simple task as well. In order to verify whether the Self-Adaptive algorithms constitute a solid option in the "real world", they were compared to the basic approach for a number of non-academic programs. In [208], test cases were generated with the basic approach for several programs taken from the book "Numerical Recipes in C. The Art of Scientific Computing." [193].

Thus, up to 16 instances that showed different levels of difficulty for the basic approach were chosen from this study, and the Self-Adaptive alternative was applied to them.

Apropos the parameters for the basic approach in [208], the EDA applied was TREE. The population consisted of 100 individuals, and the stopping criterion was reaching a maximum of 100 generations. The rest of the parameters in the EDA were the same as in Section 6.5.1. Additionaly, the test case generation was halted as soon as a limit of 100000 inputs was detected.

In the experiments with the Self-Adaptive approach, the EDA took the parameter values previously described, with two exceptions. As explained in Section 6.3.1, the EDA's population size is fixed to be twice the length of the individual. Moreover, in order to make a fair comparison, instead of using the Kullback-Leibler divergence based stopping criterion, a maximum number of generations equal to the population size was set. Again, the whole process was forced to terminate as soon as the generation of 100000 inputs was detected. In all the programs, the parameters of an input were integers or real numbers. Tentative values were adopted for the number of bits used to represent the initial and the final search regions, i.e. 5 and 10 bits for integers, and 5 and 7 bits for real parameters.

The experiments were conducted for MOA and SOA, with τ ranging from 1 to 5. Table 6.11 presents the results of the best τ for each algorithm, together with the values of the basic approach. The outstanding values are highlighted in gray.

program	basic		MOA		SOA	
	%	#	%	#	%	#
`bessj`	100	220	100	21	100	45
`bnldev`	80.77	54100	84.62	100007	84.72	100018
`caldat`	75	20100	87.5	1550	87.5	1481
`cyfun`	75	40100	75	100009	75	100011
`factln`	87.5	10330	87.5	1543	87.5	1477
`fit`	100	3760	100	101	100	101
`flmoon`	98.33	2530	100	29	100	29
`gasdev`	75	10100	75	1541	75	1476
`irbit2`	50	10100	50	1511	50	1476
`kendl1`	100	100	100	61	100	61
`laguer`	100	3590	100	2149	100	2185
`ran1`	66.67	20100	66.67	4730	66.67	3649
`ratint`	100	330	100	163	100	74
`sncndn`	93.75	10100	93.75	3089	93.75	3021
`tred2`	100	240	100	61	100	61
`tridag`	91.25	10790	100	157	100	157

Table 6.11: Results of the basic approach, MOA and SOA on real world programs.

In all the programs but one, MOA or SOA improve the outcomes of the basic approach. In this exception (`cyfyn`), the basic generator obtained a 75% coverage and stopped at 40100 inputs. The Self-Adaptive algorithms were unable to attain a better coverage,

but they continued the search over larger regions until the maximum limit of inputs was reached. Athough this behavior results undesirable in this case, it can also be very suitable. For instance, in `bnldev`, the coverage of the basic approach is augmented and the limit of 100000 is attained once again. The other programs where the coverage is outperformed are `caldat`, `flmoon` and `tridag`. For the rest of the cases, the enhacement corresponds to the number of inputs generated.

Thus, these outcomes present the Self-Adaptive approach as a viable alternative for application in the real world. Furthermore, the results clearly support the conclusion from the previous section: the Self-Adaptive algorithms perform better than the basic approach, mainly with regard to the number of inputs generated.

6.6 Summary

In this chapter, two significant topics, when the generation of test inputs is posed as an optimization problem, have been dealt with: the objective function and search space.

The former topic has been briefly tackled by comparing a function following equation 3.1 with a function using equation 3.2, in the context of the basic approach. Experimental results have shown that the second equation improves or equals the performance of the first.

Alternatively, the issue of selecting an appropriate search space was faced by describing two new approaches, namely, MOA and SOA. In order to enhance the test case generation process, the optimization step of both alternatives departs from an initial small region which is increasingly enlarged as branches remain uncovered. The starting search space is defined upon heuristic information from the program. More precisely, two options could be adopted: the application of a set of rules concerning the source code's static information, or using a heuristic procedure based on dynamic information, which consisted of a grid search method.

The analysis of the experiments conducted revealed promising results for both approaches. First of all, the search over different regions allows for the achievement of the highest coverage values, which is a primary performance measurement. Apropos the two heuristic strategies to obtain the initial region, it was concluded that the static option makes a difference and can at least improve the efficiency of the approach in terms of the number of inputs generated. Instead, the dynamic heuristic showed to be more unstable. The τ parameter of the method did not provide a relevant influence on the best values.

Comparing the performance of the MOA and SOA algorithms, in general terms, no significant difference was found between them. Additionally, the algorithms were compared to the basic approach. With the exception of the inferior results in one test program, the former outperformed the latter with statistical evidence. Moreover, this improvement

over the basic generator repeated for a number of "real-world" programs, presenting the Self-Adaptive strategy as a solid alternative.

7 Conclusions

In this chapter, the main contributions exposed throughout the dissertation are summarized. Conclusions arising from the work developed in previous chapters are included and, additionally, directions for future lines of research are suggested.

7.1 Contributions

Among the issues related to software testing, the automatic generation of the input cases to be applied to the program under test is especially relevant and difficult. A common strategy for tackling this task consists of creating inputs that fulfill an adequacy criterion based on the program structure. The present dissertation has addressed the test data generation problem, focusing on branch testing, a mandatory criterion nowadays. This task has been posed as a set of optimization problems to be solved. Then, the three significant elements related to each optimization problem have been studied, that is, the optimization method, the search space and the objective function. Among these, special attention has been paid to the optimization technique. After overviewing EDAs and SS, their application to this problem has been studied. Apropos the two other elements, since research in the field is being active for the objective function, the search space topic has been emphasized.

More specifically, the contributions of the present work may be summarized as follows:

- An overview of EDAs and SS, two modern metaheuristic techniques currently deserving the attention of the EAs community.
- In some optimization methods, there is a lack of works dealing with real-world problems, which is an important aspect for uncovering their limitations or knowing whether they represent a practical alternative. Here, EDAs and SS have been applied to the generation of test inputs for branch coverage, a demanding real-world problem.
- The application of EDAs has been studied in the context of an iterative two-step process. In the first step, a branch is chosen as the objective and, in the second, the corresponding optimization problem is tackled through an EDA. Three types

of EDAs have been evaluated empirically: those where the probabilistic model assumes problem variables are conditionally independent (UMDA and PBIL), algorithms with first order dependence probability distributions (MIMIC and TREE), and EDAs where the model makes no restriction on the dependencies between variables ($\text{EBNA}_{\text{K2+pen}}$ and EBNA_{BIC}).

- The SS methodology has been employed following the same test data generation framework as for EDAs, so enabling their comparison. Additionally, light has been shed on SS internals, which can scarcely be found in the literature to date. More precisely, the role of the improvement method in the SS algorithm has been dealt with by studying three application options. Namely, these are: using improvement in the classical way (*Improve After*), i.e. after diversification or combination, or just improving the solutions to enter in the reference set (*Improve Before*), or not employing improvement at all. Such alternatives have been analyzed empirically. Moreover, for each alternative, the weight of each SS method has been captured during the search.

- Both, EDAs and SS, have been combined to solve the test data generation. A collaborative scheme has been developed where, firstly, the EDAs based approach is applied and, once it has finished, the SS based generator is used over the remaining uncovered branches.

- Regarding the objective function, an advanced formulation (equation 3.2) has been discussed and compared with a basic function (equation 3.1) through experimentation.

- The issue of selecting an appropriate search space has been explored by developing a Self-Adaptive strategy that seeks a promising feasible region. Two algorithms conforming to this strategy have been described: MOA and SOA. The underlying idea in MOA is to apply the EDAs based framework over widening regions. By contrast, in SOA, the basic framework is used once, though the EDA executes over increasingly augmented regions. In both approaches, the initial search space is defined upon static or dynamic heuristic information from the source code of the program. Additionally, parameters of the EDA are made self-adaptive. Population size is set to twice the length of the individual and, for the stopping criterion, a novel rule based on the Kullback-Leibler divergence from the estimated to the empirical probability distribution is proposed.

7.2 Conclusions

The main general conclusion that can be drawn from the dissertation is that treatment of the test data generation from an optimization point of view proves to be successful.

To sum up, the following ideas collect the major conclusions from the work developed:

- Considering the optimization method, EDAs as well as SS are solid options for solving the test data generation. Furthermore, they are able to improve the results achieved by other methods. To be precise, upcoming concepts may be inferred from their application:
  - In the EDAs based approach, the coverage attained was 100% in all the experimental programs and, excepting a few cases, the number of inputs generated was clearly lower than in other works based on GAs. Among the different EDAs, algorithms using nontrivial probabilistic models seem to be a promising alternative. A ranking based on statistical tests was developed to identify the best algorithms; TREE and $\text{EBNA}_{\text{K2+pen}}$ showed the best overall performance. The capability of these EDAs for expressing the dependencies between problem variables could be a key point, as such dependencies usually exist when trying to cover a particular branch.
  - The SS based approach shows competitive with regard to the EDAs test data generator: in three of the seven test programs, SS improved the results of EDAs with statistical evidence, and no differences where found in another program. Concerning the influence of the improvement method, it may be concluded that, despite being optional, this element plays a main role in the SS methodology. The weight of improvement is reflected in the number of solutions generated and the number of optima found during the process. The *Improve Before* option proposed obtained statistically significant better results than the classical strategy in three of the seven test programs, thus presenting as an interesting alternative. Clearly, the worst performance was obtained if no improvement method is employed. Moreover, the behavior of other SS methods depends on the way improvement is used. In fact, a common observation to the three cases of study is the unability of the combination method to reach an important number of high quality solutions by itself. So, in this context and according to our experiments, in SS, the two prominent methods for optima attainment are improvement and diversification.
  - The empirical comparison of the EDA-SS collaborative approach with the two other points at the former as a method lying in-between the latter, from the point of view of performance.
- The evaluation of the objective function confirmed that the function defined according to equation 3.2 outperforms or equals that of 3.1.
- Apropos the study of the search space selection, this issue shows itself highly relevant to reach improved results. Although no significant difference was appreciated

between MOA and SOA approaches, the results of the basic EDAs test data generator were clearly enhanced, with the exception of one test program. The outcomes of the experiments reveal that the search over different regions allows for the achievement of the highest coverage values. For the two heuristic strategies to obtain the initial region, it was concluded that the static information based option makes a difference and can at least improve the efficiency of the approach in terms of the number of inputs generated. Instead, the dynamic heuristic showed to be more unstable.

7.3 Future Work

Undoubtedly, much research is to be undertaken in the area of optimization and, more exactly, on metaheuristics such as EDAs and SS. Just to name a few ideas, theoretical works on their behavior, parallel designs, new algorithms for EDAs, advanced methods for SS, or stopping criteria. Progress in these (and many other) topics are important for better understanding of such techniques and, ultimately, to yield wiser applications in the real world. Though this must be borne in mind, we focus next on a number of hints for future lines of work which might be interesting in our particular context.

- In the field of EDAs, a relevant topic is the selection of an appropriate algorithm for a given problem. Several works have already been developed, suggesting that simple EDAs (e.g. UMDA) are more limited in finding high quality solutions than complex EDAs (like TREE or EBNA) [259], or that the probabilistic model should capture the interactions between the objective function variables [26]. To some extent, results obtained in the dissertation conform to these studies, however, further research is needed.

- Considering SS, very few publications have been devoted to the internals of its operation. Yet we have raised a slight contribution on this matter, intensive efforts should be addressed towards the effect of each SS method during the search process. This would help in the design of the adequate SS algorithm for a given problem.

- Additional work can be conducted on the EDAs based test data generator. For instance, an appealing option is employing different EDAs for each branch to be covered, instead of a fixed one. Notice results on the selection of an appropriate EDA would be useful here.

- Concerning the EDA-SS collaborative approach, the proper balance between the parameter values of each generator should be studied with the purpose of obtaining the maximum benefit. Moreover, other forms of the collaborative scheme could be considered; for example, a SS-EDA combination.

- In the Self-Adaptive approach, several elements can be further studied. In the disappointing experimental results, almost all the initial regions where created with the dynamic information heuristic. Since the static strategy behaves superiorly, a way to enhance the response of the approach could be to expand the set of heuristic rules. On the other hand, in order to make the approach more flexible, another interesting line of future work is the elicitation of an α value for the stopping criterion of the EDA, which takes into account the size of the search space.

- The general scheme (Figure 3.5) followed throughout this dissertation owns two elements apart from the optimization phase, namely, the branch selection step and the stopping criterion. So, attention may be paid to them as well. For instance, an interesting option is to deal with the selection step, which determines the order for solving the optimization problems. Advantage may be taken from ideas that have been developed for reducing the number of branches to be covered through control flow graph analysis [140].

- Finally, we propose to extend the test data generation problem to the application of other fields in Artificial Intelligence. More precisely, in [208], a new line of work was opened with the application of Data Mining techniques in this context. The underlying idea was to study the capability of software complexity metrics to predict the performance of the EDAs based test data generator. This is a worthwhile issue, as it is a first step towards the prediction of the most desirable approach for a given program.

8 Bibliography

[1] C. W. Ahn and R. S. Ramakrishna. Elitism-based compact Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, 2003.

[2] C. W. Ahn, R. S. Ramakrishna, and D. E. Goldberg. Real-coded Bayesian optimization algorithm. In Lozano et al. [137], pages 51–73.

[3] E. Alba and J. F. Chicano. Software testing with evolutionary strategies. In N. Guelfi and A. Savidis, editors, *Proceedings of the Second International Workshop on Software Engineering Techniques*, pages 50–65. Springer-Verlag, 2005.

[4] E. Alba-Cabrera, R. Santana, A. Ochoa-Rodriguez, and M. Lazo-Cortés. Finding typical testors by using an evolutionary strategy. In F. Muge, M. Piedade, and R. Caldas Pinto, editors, *Proceedings of the 5th Ibero American Symposium on Pattern Recognition*, pages 267–278, Lisbon, Portugal, 2000.

[5] H. Aytug and G. J. Koehler. New stopping criterion for genetic algorithms. *European Journal of Operational Research*, 126(3):662–674, 2000.

[6] T. Bäck. Self-adaptation. In Bäck et al. [9], pages 188–211.

[7] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[8] T. Bäck, D. B. Fogel, and T. Michalewicz. *Evolutionary Computation 1. Basic Algorithms and Operators.* Institute of Physics Publishing, Bristol, UK, 2000.

[9] T. Bäck, D. B. Fogel, and T. Michalewicz, editors. *Evolutionary Computation 2. Advanced Algorithms and Operators.* Institute of Physics Publishing, Bristol, UK, 2000.

[10] J. Baker. Reducing bias and inefficiency in the selection algorithm. In J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 14–21. Lawrence Erlbaum Associates, 1987.

[11] S. Baluja. Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon Report, CMU-CS-94-163, 1994.

[12] S. Baluja and R. Caruana. Removing the genetics from the standard Genetic Algorithms. Technical report, Carnegie Mellon Report, CMU-CS-95-141, 1995.

[13] S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Proceedings of the 15th National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 1998, IAAI 1998, Wisconsin, USA, July 26-30, 1998*, pages 469–476. AAAI Press–The MIT Press, 1998.

[14] W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, Orlando, Florida, USA, July 13-17, 1999*. Morgan Kaufmann, 1999.

[15] A. Baresel and H. Sthamer. Evolutionary testing of flag conditions. In Cantú-Paz et al. [37], pages 2442–2454.

[16] A. Baresel, H. Sthamer, and M. Schmidt. Fitness function design to improve evolutionary structural testing. In Langdon et al. [127], pages 1329–1336.

[17] B. Beizer. *Software Testing Techniques*. Van Nostrand Rheinhold, New York, 1990.

[18] B. Beizer. *Black-Box Testing Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, New York, 1995.

[19] E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Solving graph matching with EDAs using a permutation–based representation. In Larrañaga and Lozano [130], pages 243–265.

[20] V. Berzins and Luqi. *Software Engineering with Abstractions*. Addison-Wesley, Reading, MA, 1991.

[21] H. G. Beyer, U. M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llorà, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J. P. Watson, and E. Zitzler, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005, Washington DC, USA, June 25-29, 2005*. ACM, 2005.

[22] S. Blackmore. *The Meme Machine*. Oxford University Press, Oxford, UK, 1999.

[23] R. Blanco, I. Inza, and P. Larrañaga. Learning Bayesian networks in the space of structures by Estimation of Distribution Algorithms. *International Journal of Intelligent Systems*, 18(2):2205–220, 2003.

[24] B. W. Boehm. *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[25] P. J. Boland, S. F. Sekirkin, and H. Singh. Theoretical and practical challenges in software reliability and testing. In B. H. Lindqvist and K. A. Doksum, editors, *Mathematical and Statistical Methods in Reliability.* World Scientific Publishers, New Jersey, 2003.

[26] P. A. N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In Banzhaf et al. [14], pages 60–67.

[27] L. Bottaci. Predicate expression cost functions to guide evolutionary search for test data. In Cantú-Paz et al. [37], pages 2455–2464.

[28] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, Reading, MA, 1995.

[29] D. F. Brown, A. B. Garmendia-Doval, and J. A. W. McCall. Markov random field modelling of royal road Genetic Algorithms. In P. Collet, C. Fonlupt, J. K. Hao, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2001.

[30] P. M. S. Bueno and M. Jino. Automatic test data generation for program paths using genetic algorithms. *International Journal of Software Engineering and Knowledge Engineering*, 12(6):691–709, 2002.

[31] W. Buntine. Theory refinement in Bayesian networks. In B. D'Ambrosio and P. Smets, editors, *UAI*, pages 52–60. Morgan Kaufmann, 1991.

[32] E. K. Burke and G. Kendall, editors. *Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques.* Springer, New York, 2005.

[33] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, 21(4):397–414, 2001.

[34] E. Cantú-Paz. *Efficient and accurate parallel Genetic Algorithms.* Kluwer Academic Publishers, 2000.

[35] E. Cantú-Paz. Feature subset selection by Estimation of Distribution Algorithms. In Langdon et al. [128], pages 303–310.

[36] E. Cantú-Paz. Pruning Neural Networks with Distribution Estimation Algorithms. In Cantú-Paz et al. [38], pages 790–800.

[37] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. O'Reilly, H. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors. *Proceedings of the Genetic and Evolutionary Computation Conference*, Berlin, 2003. Springer-Verlag.

[38] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. M. O'Reilly, H. G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003, Part I, Chicago, IL, USA, July 12-16, 2003*, volume 2723 of *Lecture Notes in Computer Science*. Springer, 2003.

[39] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. M O'Reilly, H. G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003, Part II, Chicago, IL, USA, July 12-16, 2003*, volume 2724 of *Lecture Notes in Computer Science*. Springer, 2003.

[40] E. Castillo, J. M. Gutiérrez, and A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer-Verlag, New York, 1997.

[41] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[42] L. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, 2(3):215–222, 1976.

[43] Y. Collette and P. Siarry. *Multiobjective Optimization. Principles and Case Studies*. Springer-Verlag, Berlin, 2003.

[44] G. F. Cooper and E. A. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[45] D. Corne, Z. Michalewicz, B. McKay, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, G. Raidl, K. C. Tan, and A. Zalzala, editors. *Proceedings of the 2005 Congress on Evolutionary Computation, CEC-2005, Edinburgh, U.K., September 2-5, 2005*. IEEE Press, 2005.

[46] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza-Reorda. Exploiting competing subpopulations for automatic generation of test sequences for digital circuits. In Voigt et al. [242], pages 792–800.

[47] C. Cotta, E. Alba, R. Sagarna, and P. Larrañaga. Adjusting weights in artificial neural networks using evolutionary algorithms. In Larrañaga and Lozano [130], pages 357–373.

[48] C. Gomes da Silva, J. Clímaco, and J. Figueira. A scatter search method for bi-criteria {0,1}-knapsack problems. *European Journal of Operational Research*, 169(2):373–391, 2006.

[49] A. L. Dake and R. M. Keller. Data flow program paths. *IEEE Computer*, 15(2):26–41, 1982.

[50] C. Darwin. *The Origin of Species*. John Murray, 1859.

[51] D. Dasgupta and Z. Michalewicz. *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, Berlin, 1997.

[52] M. Davis and E. J. Weyuker. A formal notion of program-based test data adequacy. *Information and Control*, 56(1–2):52–71, 1983.

[53] J. S. De Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. Jordan M. Mozer and Th. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, 1997.

[54] L. M. De Campos, J. A. Gámez, P. Larrañaga, S. Moral, and T. Romero. Partial abductive inference in Bayesian networks: an empirical comparison between GAs and EDAs. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 323–341. Kluwer Academic Publishers, 2002.

[55] K. Deb, R. Poli, W. Banzhaf, H. G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2004, Part II, Seattle, WA, USA, June 26-30, 2004*, volume 3103 of *Lecture Notes in Computer Science*. Springer, 2004.

[56] R. Demillo and A. Offut. Experimental results from an automatic test case generator. *ACM Transactions on Software Engineering and Methodology*, 2(2):109–127, 1993.

[57] E. Díaz, J. Tuya, and R. Blanco. Automated software testing using a metaheuristic technique based on tabu search. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pages 310–313. IEEE CS Press, 2003.

[58] S. Dick and A. Kandel. Data mining with resampling in software metrics databases. In M. Last, A. Kandel, and H. Bunke, editors, *Artificial Intelligence Methods in Software Testing*, pages 175–208. World Scientific Publishing, Singapore, 2004.

[59] S. H. Dick. *Computational Intelligence in Software Quality Assurance*. PhD thesis, University of South Florida, Tampa, Florida, USA, 2002.

[60] E. Dijkstra. *A discipline of programming*. Prentice-Hall, New Jersey, 1976.

[61] M. Dorigo. Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Italy, 1991.

[62] S. Droste. Not all linear functions are equally difficult for the compact Genetic Algorithm. In Beyer et al. [21], pages 679–686.

[63] R. H. Dunn. *Software Quality. Concepts and Plans*. Prentice-Hall, New Jersey, 1990.

[64] J. W. Duran and S. C. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering*, 10(4):438–444, 1984.

[65] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–139, 1999.

[66] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

[67] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering*. Pearson Education Limited, London, 2003.

[68] R. Etxeberria and P. Larrañaga. Global optimization with Bayesian networks. In Ochoa et al. [169], pages 332–339.

[69] N. E. Fenton. The structural complexity of flowgraphs. In Y. Alavy, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall, editors, *Graph Theory with Applications to Algorithms and Computer Science*, pages 273–282. John Wiley & Sons, New York, 1985.

[70] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[71] R. Ferguson and B. Korel. The chaining approach for software test data generation. *IEEE Transactions on Software Engineering*, 5(1):63–86, 1996.

[72] M. J. Flores, J. A. Gámez, and J. M. Puerta. Learning linguistic fuzzy rules by using estimation of distribution algorithms as the search engine in the cor methodology. In Lozano et al. [137], pages 259–280.

[73] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through a simulation of evolution. In A. Callahan, M. Maxfield, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems*, pages 131–156. Spartan, Washington DC, 1965.

[74] International Organization for Standardization. *Software Engineering - Product Quality - Part1: Quality Model.* ISO/IEC 9126-1:2001(E). ISO, Geneva, Switzerland, 2001.

[75] P. Frankl, D. Hamlet, B. LittleWood, and L. Strigini. Choosing a testing method to deliver reliability. *IEEE Transactions on Software Engineering*, 24(8):586–601, 1998.

[76] P. Frankl and E. J. Weyuker. A formal analysis of the fault-detecting ability of testing methods. *IEEE Transactions on Software Engineering*, 19(3):202–213, 1993.

[77] P. Frankl and E. J. Weyuker. Provable improvements on branch testing. *IEEE Transactions on Software Engineering*, 19(10):962–975, 1993.

[78] M. A. Friedman and J. M. Voas. *Software Assessment: Reliability, Safety, Testability.* John Wiley & Sons, New York, 1995.

[79] P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[80] J. C. Gallagher and S. Vigraham. A Modified Compact Genetic Algorithm For The Intrinsic Evolution of continuous time recurrent neural networks. In Langdon et al. [128], pages 163–170.

[81] M. R. Gallagher. *Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling.* PhD thesis, University of Queensland, 2000.

[82] Y. Gao and J. Culberson. Space Complexity of Estimation of Distribution Algorithms. *Evolutionary Computation*, 13(1):125–143, 2005.

[83] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Company, New York, 1979.

[84] F. Glover. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13:879–919, 1965.

[85] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 297–316. McGraw-Hill, Cambridge, 1999.

[86] F. Glover and G. Kochenberger. *Handbook of Metaheuristics.* Kluwer Academic Publishers, Boston, 2003.

[87] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, Boston, 1997.

[88] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[89] C. González, J. A. Lozano, and P. Larrañaga. Analyzing the population based incremental learning algorithm by means of discrete dynamical systems. *Complex Systems*, 12(4):465–479, 2000.

[90] C. González, J. D. Rodríguez, J. A. Lozano, and P. Larrañaga. Analysis of the Univariate Marginal Distribution Algorithm modeled by Markov chains. In J. Mira and J. R. Álvarez, editors, *IWANN 2003*, volume 2686 of *Lecture Notes in Computer Science*, pages 510–517. Springer, 2003.

[91] T. Gosling. The simple supply chain model and evolutionary computation. In Sarker et al. [215], pages 2322–2329.

[92] E. L. Grant and R. S. Leavenworth. *Statistical Quality Control*. McGraw-Hill, New York, 1996.

[93] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[94] W. Gutjahr. Importance sampling of test cases in markovian software usage models. *Probability in the Engineering and Informational Sciences*, 11:19–36, 1997.

[95] H. Handa. Hybridization of Estimation of Distribution Algorithms with a repair method for solving constraint satisfaction problems. In Cantú-Paz et al. [38], pages 991–1002.

[96] H. Handa. Estimation of Distribution Algorithms with mutation. In G. R. Raidl and J. Gottlieb, editors, *EvoCOP*, volume 3448 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2005.

[97] G. Harik. Linkage learning in via probabilistic modeling in the EcGA. Technical Report 99010, IlliGAL, 1999.

[98] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact Genetic Algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

[99] M. Harman, C. Fox, R. Hierons, L. Hu, S. Danicic, and J. Wegener. Vada: A transformation-based system for variable dependence analysis. In *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation*, pages 55–64, Los Alamitos, CA, 2002. IEEE CS Press.

[100] M. Harman, L. Hu, R. Hierons, A. Baresel, and H. Sthamer. Improving evolutionary testing by flag removal. In Langdon et al. [127], pages 1351–1358.

[101] M. S. Hecht. *Flow Analysis of Computer Programs.* Elsevier Science, New York, 1977.

[102] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[103] M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In J. F. Lemmer and L. N. Kanal, editors, *Proceedings of the Second Conference on Uncertainty in Artificial Intelligence*, pages 149–163, Amsterdam, 1988. North-Holland.

[104] I. Hermadi and M. A. Ahmed. Genetic algorithm based test data generator. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 85–91, NJ, 2003. IEEE CS Press.

[105] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for the behaviour analysis. *Artificial Intelligence Review*, 12:265–319, 1998.

[106] M. Höhfeld and G. Rudolph. *Towards a Theory of Population-Based Incremental Learning.* International Conference on Evolutionary Computation, Indianapolis, USA, April 13-16, 1997.

[107] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[108] Y. Hong, Q. Ren, and J. Zeng. Adaptive population size for Univariate Marginal Distribution Algorithm. In Corne et al. [45], pages 1396–1402.

[109] Y. Hong, Q. Ren, and J. Zeng. Genetic drift in Univariate Marginal Distribution Algorithm. In Beyer et al. [21], pages 745–746.

[110] Y. Hong, Q. Ren, and J. Zeng. Optimization of noisy fitness functions with Univariate Marginal Distribution Algorithm. In Corne et al. [45], pages 1410–1417.

[111] I. Inza, P. Larrañaga, R. Etxeberria, and B. Sierra. Feature subset selection by bayesian networks based optimization. *Artificial Intelligence*, 123(1-2):157–184, 2000.

[112] I. Inza, P. Larrañaga, and B. Sierra. Feature Subset Selection by Estimation of Distribution Algorithms. In Larrañaga and Lozano [130], pages 269–294.

[113] I. Inza, P. Larrañaga, and B. Sierra. Estimation of Distribution Algorithms for feature Subset Selection in large dimensionality domains. In H. Abbass, R. Sarker, and C. Newton, editors, *Data Mining: A Heuristic Approach*, pages 97–116. IDEA Group Publishing, 2002.

[114] C. Jones. *Software Quality: Analysis and Guidelines for Success.* International Thompson Computer Press, New York, 1997.

[115] C. Jones. *Software Assessments, Benchmarks and Best Practices.* Addison-Wesley, New York, 2000.

[116] C. Kaner, J. Bach, and B. Pettichord. *Lessons Learned in Software Testing.* John Wiley & Sons, New York, 2001.

[117] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3(1):77–112, 2004.

[118] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3(3):355–356, 2004.

[119] B. W. Kernighan, D. Ritchie, and D. M. Ritchie. *The C Prgramming Language.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[120] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[121] B. Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, 16(8):870–879, 1990.

[122] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1992.

[123] M. Laguna. Scatter search. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 183–193. Oxford University Press, 2002.

[124] M. Laguna and V. A. Armentano. Lessons from applying and experimenting with scatter search. In Rego and Alidaee [197], pages 229–246.

[125] M. Laguna and R. Martí. *Scatter Search. Methodology and Implementations in C.* Kluwer Academic Publishers, Norwell, MA, 2003.

[126] M. Laguna and R. Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33:235–255, 2005.

[127] W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D.Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors. *Proceedings of the Genetic and Evolutionary Computation Conference*, San Mateo, CA, 2002. Morgan Kaufmann.

[128] W. B. Langdon, E. Cantú-Paz, K. E. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. K. Burke, and N. Jonoska, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, New York, USA, July 9-13, 2002*. Morgan Kaufmann, 2002.

[129] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In C. Boutilier and M. Goldszmidt, editors, *UAI*, pages 343–352. Morgan Kaufmann, 2000.

[130] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston, 2002.

[131] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[132] N. Leverson. *Safeware: System Safety and Computers*. Addison-Wesley, Boston, 1995.

[133] C. F. Lima, K. Sastry, D. E. Goldberg, and F. G. Lobo. Combining competent crossover and mutation operators: a probabilistic model building approach. In Beyer et al. [21], pages 735–742.

[134] J. Lin and P. Yeh. Automatic test data generation for path testing using gas. *Information Sciences*, 131:47–64, 2001.

[135] X. Llorà and D. E. Goldberg. Wise breeding GA via machine learning techniques for function optimization. In Cantú-Paz et al. [38], pages 1172–1183.

[136] J. A. Lozano, P. Larrañaga, M. Graña, and F. X. Albizuri. Genetic algorithms: bridging the convergence gap. *Theoretical Computer Science*, 229:11–22, 1999.

[137] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a new Evolutionary Computation: Advances in Estimation of Distribution Algorithms*. Springer-Verlag, The Netherlands, 2006.

[138] J. A. Lozano and A. Mendiburu. Solving job schedulling with Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 231–242. Kluwer Academic Publishers, 2002.

[139] J. A. Lozano, R. Sagarna, and P. Larrañaga. Parallel estimation of distribution algorithms. In Larrañaga and Lozano [130], pages 125–142.

[140] M. Marré and A. Bertolino. Using spanning sets for coverage testing. *IEEE Transactions on Software Engineering*, 29(11):974–984, 2003.

[141] R. Martí, M. Laguna, and V. Campos. Scatter search vs. genetic algorithms. an experimental evaluation with permutation problems. In Rego and Alidaee [197], pages 263–282.

[142] G. McGraw, C. Michael, and M. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, 27(12):1085–1110, 2001.

[143] P. McMinn. Search-based software test data generation: a survey. *Software Testing Verification and Reliability*, 14(2):105–156, 2004.

[144] A. Mendiburu, J. A. Lozano, and J. Miguel-Alonso. Parallel implementation of EDAs based on probabilistic graphical models. *IEEE Transactions on Evolutionary Computation*, 9(4):406–423, 2005.

[145] A. Mendiburu, J. Miguel-Alonso, J. A. Lozano, M. Ostra, and C. Ubide. Parallel and multi-objective EDAs to create multivariate calibration models for quantitative chemical applications. In T. Skie and C. S. Yang, editors, *ICPP Workshops*, pages 596–603. IEEE Computer Society, 2005.

[146] W. Miller and D. Spooner. Automatic generating of floating-point test data. *IEEE Transactions on Software Engineering*, 2(3):223–226, 1976.

[147] N. Mladenović. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[148] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan-Kaufman, San Mateo, 1991.

[149] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1998.

[150] H. Mühlenbein and T. Mahnig. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7:19–32, 1999.

[151] H. Mühlenbein and T. Mahnig. The factorized distribution algorithm for additively decomposed functions. In Ochoa et al. [169], pages 301–313.

[152] H. Mühlenbein and T. Mahnig. Evolutionary algorithms: From recombination to search distributions. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, pages 135–173. Springer, Berlin, 2001.

[153] H. Mühlenbein and T. Mahnig. Mathematical analysis of evolutionary algorithms for optimization. In *Proceedings of the Third International Symposium on Adaptive Systems*, pages 166–185, La Havana, Cuba, 2001.

[154] H. Mühlenbein and T. Mahnig. Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal of Approximate Reasoning*, 31(3):157–192, 2002.

[155] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

[156] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In Voigt et al. [242], pages 178–187.

[157] J. D. Musa. Operational profiles in software reliability engineering. *IEEE Software*, 10(2):14–32, 1993.

[158] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[159] S. C. Ntafos. A comparison of some structural testing strategy. *IEEE Transactions on Software Engineering*, 14(6):868–874, 1988.

[160] J. Ocenasek. Entropy-based convergence measurement in discrete estimation of distribution algorithms. In Lozano et al. [137], pages 39–50.

[161] J. Ocenasek, S. Kern, N. Hansen, and P. Koumoutsakos. A mixed Bayesian optimization algorithm with variance adaptation. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H. P. Schwefel, editors, *PPSN*, volume 3242 of *Lecture Notes in Computer Science*, pages 352–361. Springer, 2004.

[162] J. Ocenasek and J. Schwarz. The parallel Bayesian optimization algorithm. In *Proceedings of the European Symposium on Computational Intelligence*, pages 61–67, 2000.

[163] J. Ocenasek and J. Schwarz. The distributed Bayesian optimization algorithm for combinatorial optimization. In *EUROGEN - Evolutionary Methods for Design, Optimisation and Control, CIMNE*, pages 115–120, 2001.

[164] J. Ocenasek and J. Schwarz. Estimation of distribution algorithm for mixed continuous-discrete optimization problems. In *2nd Euro-International Symposium on Computational Intelligence*, pages 227–232. IOS Press, Kosice, Slowakia, 2002.

[165] J. Ocenasek, J. Schwarz, and M. Pelikan. Design of multithreaded Estimation of Distribution Algorithms. In Cantú-Paz et al. [39], pages 1247–1258.

[166] A. Ochoa, H. Mühlenbein, and M. Soto. Factorized Distribution Algorithm using Bayesian networks. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 212–215, 2000.

[167] A. Ochoa, H. Mühlenbein, and M. Soto. A Factorized Distribution Algorithm using single connected Bayesian networks. In Schoenauer et al. [220], pages 787–796.

[168] A. Ochoa, M. Soto, R. Santana, J. Madera, and N. Jorge. The factorized distribution algorithm and the junction tree: A learning perspective. In Ochoa et al. [169], pages 368–377.

[169] A. Ochoa, M. R. Soto, and R. Santana, editors. *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, Habana, Cuba, 1999.

[170] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Standard 610.12-1990. IEEE, New York, 1990.

[171] Institute of Electrical and Electronics Engineers. *IEEE Standard for a Software Quality Metrics Methodology.* IEEE Standard 1061-1992. IEEE, New York, 1992.

[172] J. Offutt, Z. Jin, and J. Pan. The dynamic domain reduction procedure for test data generation. *Software - Practice and Experience*, 29(2):167–193, 1999.

[173] J. Offutt, S. Liu, A. Abduzarik, and P. Ammann. Generating test data from state-based specifications. *Software Testing, Verification and Reliability*, 13:25–53, 2003.

[174] M. Ould. Testing- a challenge to method and tool developers. *Software Engineering Journal*, 6(2):59–64, 1991.

[175] R. Pargas, M. Harrold, and R. Peck. Test-data generation using genetic algorithms. *Journal of Software Testing, Verification and Reliability*, 9(4):263–282, 1999.

[176] T. K. Paul and H. Iba. Linear and combinatorial optimizations by Estimation of Distribution Algorithms. In *Proceedings of the 9th MPS Symposium on Evolutionary Computation*, pages 99–106, 2003.

[177] T. K. Paul and H. Iba. Reinforcement Learning Estimation of Distribution Algorithm. In Cantú-Paz et al. [39], pages 1259–1270.

[178] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, New York, 1984.

[179] M. Pelikan and D. E. Goldberg. Genetic Algorithms, clustering, and the breaking of symmetry. In Schoenauer et al. [220], pages 385–394.

[180] M. Pelikan and D. E. Goldberg. Hierarchical problem solving and the Bayesian optimization algorithm. In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 267–274, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[181] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent Genetic Algorithms. In Spector et al. [230], pages 511–518.

[182] M. Pelikan and D. E. Goldberg. Hierarchical BOA solves Ising spin glasses and MAXSAT. In Cantú-Paz et al. [39], pages 1271–1282.

[183] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In Banzhaf et al. [14], pages 525–532.

[184] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation and Bayesian networks. *Evolutionary Computation*, 8(3):311–340, 2000.

[185] M. Pelikan, D. E. Goldberg, and K. Sastry. Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor. In Spector et al. [230], pages 519–526.

[186] M. Pelikan and T. K. Lin. Parameter-less hierarchical BOA. In Deb et al. [55], pages 24–35.

[187] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chandhory, editors, *Advances in Soft Computing-Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[188] M. Pelikan and K. Sastry. Fitness inheritance in the Bayesian optimization algorithm. In Deb et al. [55], pages 48–59.

[189] J. M. Peña, J. A. Lozano, and P. Larrañaga. Globally Multimodal Problem Optimization Via an Estimation of Distribution Algorithm Based on Unsupervised Learning of Bayesian Networks. *Evolutionary Computation*, 13(1):43–66, 2005.

[190] R. Petrasch. The definition of software quality: A practical approach. In *Proceedings of the Tenth International Symposium on Software Reliability Engineering*, pages 33–34, Boca Raton, Florida, 1999. IEEE CS Press.

[191] H. Pham. *Software Reliability*. Springer-Verlag, Singapore, 2000.

[192] H. Pohlheim and J. Wegener. Testing the temporal behavior of real-time software modules using extended evolutionary algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Second Genetic and Evolutionary Computation Conference*, pages 1795–1802, San Francisco, CA, 1999. Morgan Kaufmann.

[193] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, New York, 1988.

[194] R. Rastegar and M. R. Meybodi. A new estimation of distribution algorithm based on learning automata. In Corne et al. [45], pages 1982–1987.

[195] I. Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.

[196] C. R. Reeves. Modern heuristic techniques. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 1–25. John Wiley & Sons, New York, 1996.

[197] C. Rego and B. Alidaee, editors. *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, Norwell MA, 2005.

[198] J. Rissanen. Modeling by shortest data description. Automatica , 465–471, 1978.

[199] V. Robles, P. de Miguel, and P. Larrañaga. Solving the Traveling Salesman Problem with Estimation of Distribution Algorithms. In Larrañaga and Lozano [130], pages 211–229.

[200] T. Romero, P. Larrañaga, and B. Sierra. Learning Bayesian networks in the space of orderings with Estimation of Distribution Algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(4):607–625, 2004.

[201] P. Ross. Hyper-heuristics. In Burke and Kendall [32], pages 529–556.

[202] J. Roure, P. Larrañaga, and R. Sangüesa. An empirical comparison between k-means, GAs and EDAs in partitional clustering. In Larrañaga and Lozano [130], pages 343–360.

[203] G. Rudolph. Global optimization by means of distributed evolution strategies. In H. P. Schwefel and R. Männer, editors, *Proceedings of the First International Conference on Parallel Problem Solving from Nature*, pages 209–213, Berlin, 1990. Springer-Verlag.

[204] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 50–54, Piscataway, NJ, 1996. IEEE CS Press.

[205] D. Ruta and B. Gabrys. Application of the evolutionary algorithms for classifier selection in multiple classifier systems with majority voting. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2001.

[206] R. Sagarna and P. Larrañaga. Solving the knapsack problem with estimation of distribution algorithms. In Larrañaga and Lozano [130], pages 195–209.

[207] R. Sagarna and J. A. Lozano. Dynamic search space transformations for software test data generation. Submitted.

[208] R. Sagarna and J. A. Lozano. Software metrics mining to predict the performance of estimation of distribution algorithms in test data generation. In C. Cotta, editor, *Data-driven Knowledge*. Springer-Verlag. In Press.

[209] R. Sagarna and J. A. Lozano. Variable search space for software testing. In *Proceedings of the IEEE International Conference on Neural Networks and Signal Processing*, pages 575–578, Nanjing, China, 2003. IEEE CS Press.

[210] R. Sagarna and J. A. Lozano. On the performance of estimation of distribution algorithms applied to software testing. *Applied Artificial Intelligence*, 19(5):457–489, 2005.

[211] R. Sagarna and J. A. Lozano. Scatter search in software testing, comparison and collaboration with estimation of distribution algorithms. *European Journal of Operational Research*, 169(2):392–412, 2006.

[212] R. Santana. Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.

[213] R. Santana and A. Ochoa. Dealing with constraints with Estimation of Distribution Algorithms: The univariate case. In Ochoa et al. [169], pages 378–384.

[214] R. Santana, F. B. Pereira, E. Costa, A. Ochoa-Rodriguez, P. Machado, A. Cardoso, and M. R. Soto. Probabilistic evolution and the Busy Beaver problem. In Whitley et al. [251], page 380.

[215] R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors. *Proceedings of the 2003 Congress on Evolutionary Computation, CEC2003, Canberra, Australia, 8-12 December, 2003.* IEEE Press, 2003.

[216] K. Sastry, H. A. Abbass, D. E. Goldberg, and D. D. Johnson. Sub-structural niching in Estimation of Distribution Algorithms. In Beyer et al. [21], pages 671–678.

[217] K. Sastry and D. E. Goldberg. On Extended Compact Genetic Algorithm. In Whitley et al. [251], pages 352–359.

[218] K. Sastry and D. E. Goldberg. Designing competent mutation operators via probabilistic model building of neighborhoods. In Deb et al. [55], pages 114–125.

[219] F. Schoen. Two phase methods for global optimization. In P. Pardalos and E. Romeijn, editors, *Handbook of Global Optimization 2: Heuristic Approaches*, pages 151–178. Kluwer Academic Publishers, The Netherlands, 2002.

[220] M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo-Guervós, and H. P. Schwefel, editors. *Proceeding of the 6th International Conference on Parallel Problem Solving from Nature, PPSN VI, Paris, France, September 18-20, 2000*, volume 1917 of *Lecture Notes in Computer Science*. Springer, 2000.

[221] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464, 1978.

[222] H. P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[223] S. Shakya, J. McCall, and D. Brown. Updating the probability vector using MRF technique for a univariate EDA. In E. Onaindia and S. Staab, editors, *STAIRS 2004, Proceedings of the Second Starting AI Researchers' Symposium*, volume 109 of *Frontiers in Artificial Intelligence and Applications*, Valencia, Spain, August 2004. IOS Press.

[224] S. Shakya, J. McCall, and D. F. Brown. Estimating the distribution in an EDA. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, editors, *Adaptive and Natural Computing Algorithms*, Springer Computer Series, pages 202–205, Coimbra, Portugal, 21-23 March 2005. Springer.

[225] J. L. Shapiro. Drift and Scaling in Estimation of Distribution Algorithms. *Evolutionary Computation*, 13(1):99–123, 2005.

[226] M. Shepperd. *Software Engineering Metrics I: Measures and Validations*. McGraw-Hill, New York, 1993.

[227] B. Sierra, E. Jiménez, I. Inza, P. Larrañaga, and J. Muruzábal. Rule induction using Estimation of Distribution Algorithms. In Larrañaga and Lozano [130], pages 313–322.

[228] J. Smith and T. C. Fogarty. Evolving software test data - ga's learn self expression. In T. C. Fogarty, editor, *Proceedings of Evolutionary Computing. AISB Workshop*, pages 346–354, Berlin, 1996. Springer-Verlag.

[229] M. Soto, A. Ochoa, S. Acid, and L. M. de Campos. Introducing the polytree aproximation of distribution algorithm. In Ochoa et al. [169], pages 360–367.

[230] L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2001, San Francisco, California, USA, July 7-11, 2001*. Morgan Kaufmann, 2001.

[231] D. Spinellis. *Code Quality. The Open Source Perspective*. Addison-Wesley, Boston, MA, 2006.

[232] P. Spirtes, C. Glymour, and R. Scheines. An algorithm for fast recovery of sparse causal graphs. *Social Science Computing Reviews*, 9:62–72, 1991.

[233] H. Sthamer. *The Automatic Generation of Software Test Data Using Genetic Algorithms*. PhD thesis, University of Glamorgan, Pontyprid, Wales, Great Britain, 1996.

[234] R. Sukthankar, S. Baluja, and J. Hancock. Multiple adaptive agents for tactical driving. *Applied Intelligence*, 9(1):7–23, 1998.

[235] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 332–349. Van Nostrand Reinhold, 1991.

[236] G. Syswerda. Simulated crossover in genetic algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 239–255. Morgan Kaufmann, 1993.

[237] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report 02-3, National Institute of Standards and Technology, Gaithersburg, MD, 2002.

[238] N. Tracey. *A Search-Based Automated Test-Data Generation Framework for Safety Critical Software*. PhD thesis, University of York, UK, 2000.

[239] N. Tracey, J. Clark, K. Mander, and J. McDermid. An automated framework for structural test-data generation. In D. Redmiles and B. Nuseibeh, editors, *Proceedings of the 13th IEEE Conference on Automated Software Engineering*, pages 285–288. IEEE CS Press, 1998.

[240] M. Tsuji, M. Munetomo, and K. Akama. Modeling dependencies of loci with string classification according to fitness differences. In Deb et al. [55], pages 246–257.

[241] S. Tsutsui. Probabilistic Model-Building Genetic Algorithms in permutation representation domain using edge histogram. In J. J. Merelo-Guervós, P. Adamidis, H. G. Beyer, J. L. Fernández-Villacañas Martín, and H. P. Schwefel, editors, *PPSN*, volume 2439 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2002.

[242] H. M. Voigt, W. Ebeling, I. Rechenberger, and H. P. Schwefel, editors. *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, Berlin, 1996. Springer-Verlag.

[243] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, 1999.

[244] S. Waser and M. J. Flynn. *Introduction to Arithmetic for Digital Systems*. Holt, Rinehort & Winston, New York, 1982.

[245] A. Watkins and E. M. Hufnagel. Evolutionary test data generation: a comparison of fitness functions. *Software - Practice and Experience*, 36:95–116, 2006.

[246] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, 2001.

[247] S. N. Weiss. What to compare when comparing data adequacy criteria. *Software Engineering Notes*, 14(6):42–49, 1989.

[248] E. J. Weyuker. Can we measure software testing effectiveness? In *Proceedings of the First International Software Metrics Symposium*, pages 100–107. IEEE CS Press, 1993.

[249] E. J. Weyuker and B. Jeng. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering*, 17(7):703–711, 1991.

[250] D. Whitley, K. Mathias, and P. Fitzforn. Delta coding: An iterative search strategy for genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84, San Mateo, CA, 1991. Morgan Kaufmann.

[251] L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H. G. Beyer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2000, Las Vegas, Nevada, USA, July 8-12, 2000*. Morgan Kaufmann, 2000.

[252] D. H. Wolpert and W. G. MacReady. No free lunch theorems for optimisation. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[253] A. H. Wright, R. Poli, C. R. Stephens, W. B. Langdon, and S. Pulavarty. An estimation of distribution algorithm based on maximum entropy. In Deb et al. [55], pages 343–354.

[254] K. Yanai and H. Iba. Estimation of distribution programming based on Bayesian network. In Sarker et al. [215], pages 1618–1625.

[255] K. Yanai and H. Iba. Program evolution by integrating EDP and GP. In K. Deb, R. Poli, W. Banzhaf, H. G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. Luca Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 774–785. Springer, 2004.

[256] S. Yang. Memory-enhanced Univariate Marginal Distribution Algorithms for Dynamic Optimization Problems. In Corne et al. [45], pages 2560–2567.

[257] S. Yang. Population-Based Incremental Learning with memory scheme for changing environments. In Beyer et al. [21], pages 711–718.

[258] S. Yang and X. Yao. Experimental study on polulation-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.

[259] Q. Zhang. On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. *IEEE Transactions on Evolutionary Computation*, 8(1):80–93, 2004.

[260] Q. Zhang and H. Mühlenbein. On global convergence of FDA with proportionate selection. In Ochoa et al. [169], pages 340–343.

[261] A. A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic Publishers, The Netherlands, 1991.