Scatter Search and Estimation of Distribution Algorithms in Software Testing: competition and cooperation

Ramón Sagarna[*], Jose A. Lozano

Department of Computer Science and Artificial Intelligence

University of the Basque Country

Paseo Manuel Lardizabal 1

20018 San Sebastián, Guipuzcoa, Spain

E-mail: {ccbsaalr,ccploalj}@si.ehu.es

Abstract

One of the main tasks software testing includes is the generation of the test cases to be used during the test. Due to its expensive cost, the automatization of this task has become one of the key issues in the area. The field of Evolutionary Testing deals with this problem by means of combinatorial optimization search techniques.

An Evolutionary Testing based approach to the automatic generation of test inputs is presented. The developed approach involves different possibilities of the usage of two heuristic optimization methods, namely, Scatter Search and Estimation of Distribution Algorithms. The possibilities comprise several pure Scatter Search options, Scatter Seach and Estimation of Distribution Algorithm hybrids, and a cooperative alternative with both methods. Several experiments were conducted in order to evaluate and compare the presented approaches with those in the literature. The analysis of the experimental results raises interesting conclusions, showing these alternatives as a promising option for tackling this problem.

Keywords: evolutionary computations, software testing, branch coverage, scatter search, estimation of distribution algorithms

1 Introduction

Testing plays a main role in the search for the required quality of software. It is the primary way used in practice to verify the correct behaviour of the produced software. In fact, usually, at least 50% of

[*]corresponding author (ccbsaalr@si.ehu.es)

the project resources are dedicated to this phase (Beizer, 1990).

A key task in testing is the generation of the input cases to be applied to the programme under test. This is a critical task, as the produced input must be adequate to the test type and its requirements. This, together with the fact that in most of the organizations this generation is performed manually, results in a large amount of resources dedicated to the test data generation step. Thus, the automatic generation of test cases is desirable and has turned into one of the main working lines in the area.

Existing methods for test data generation mainly follow three strategies: random, functional and structural. Random testing samples input domain according to a probability distribution. Due to its simplicity, a common approach is automatically generating test cases simulating a uniform distribution (Duran and Ntafos, 1984). Alternatively, in functional testing, the inputs are obtained from the programme specification. That is, they are generated taking the functional properties of the programme into account. Its automatization demands a formal specification, e.g. in (Burton, 2000) a framework orientated to a popular specification language named Z was described.

In contrast, structural testing is based on the internal structure of the programme. The source code reveals control or data flow entities such as the branches the flow of control can take from a conditional statement or the different possible usages of a variable. These entities define several adequacy criteria which generators search to fulfil by producing adequate test inputs. For instance, branch coverage is a classical criterion stating that every programme branch must be exercised. Other typical criteria are passing through every code statement (statement coverage), exercising every sequence of branches from the programme input to its output (path coverage), or covering all the pairs definition-usage for each variable (all-defs coverage). Regardless of the adequacy criterion, there is a need to know the level of completion attained by the test case generator. This is known through the coverage measurement, i.e. the percentage of structural entities exercised for a given criterion.

Generally, automated structural testing is reached by means of static or dynamic test data generation. Static methods are commonly based on symbolic execution. This technique consists on choosing an entity from a graph representing the programme structure, and building a system of inequalities by assigning symbolic values to variables affecting the entity and respecting the constraints associated to the conditions in the code. A solution to the system is an input exercising the selected entity. In (Demillo and Offut, 1993), a work using this method can be consulted. On the other hand, the underlying idea in dynamic methods is addressing the automatic generation of test data as an optimization problem. An instrumented version of the programme is constructed, that is, the programme 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 (Korel, 1990), 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.

A field which is arousing the interest of researchers is Evolutionary Testing (ET). The aim of ET is to seek for test cases employing modern heuristic combinatorial optimization techniques during the process, to be precise, the search for appropriate test inputs is transformmed into an optimization problem to be solved through these techniques. Promising results have already been attained in the field for structural adequacy criteria (Jones et al., 1996) (Wegener et al., 2001) (Sagarna and Lozano, 2003(a)).

The present work describes an ET approach for the automatic generation of test cases. The proposed method deals with the branch coverage adequacy criterion through a two-step process, selection of the next branch to exercise and search for an input reaching this branch. This search step is dealt by considering two evolutionary methods, namely, Scatter Search (SS) (Laguna and Martí, 2003) and Estimation of Distribution Algorithms (EDAs) (Larrañaga and Lozano, 2002).

SS is an emerging evolutionary search method which has been succesfully applied to several hard optimization problems (Campos et al., 2001). This technique is based on the maintenance of a low cardinality set of solutions which is updated with new solutions obtained from the combination of the members of the set.

On the other hand, EDAs are a new set of evolutionary algorithms which, instead of creating new individuals through the classical crossover and mutation operators, as occurs in Genetic Algorithms, estimate the probability distribution associated to the selected individuals and sample this distribution to create the next population. As EDAs have already been applied to the test case generation problem with excellent results (Sagarna and Lozano, 2003(a)), they constitute an adequate benchmark for comparison with SS. Moreover, they can be considered as a promising option from which SS may benefit in order to improve its performance.

Thus, the approach here exposed makes use of the optimization methods above in two different ways. On the one hand, several pure SS alternatives are presented in order to evaluate their performance and compare them with the approaches in previous works. On the other hand, several SS-EDA hybrids and cooperative alternatives with both methods are included. As far as we know, this is the first time SS is applied to the test case generation problem, whilst, to the best of our knowledge, SS and EDA have never previously been combined.

The rest of the paper is organized as follows. The next section reviews ET and pinpoints a few salient references from this field. Sections 3 and 4 make a general introduction to SS and EDAs respectively. Pure SS approaches, together with the experiments and their results analysis, are described in Section 5. Sections 6 and 7 explain the SS and EDA combining approaches, and discuss the results of the conducted experiments. Finally, obtained conclusions and future work are included.

Repeat until stopping criterion met

$E \leftarrow$ Select entity to exercise

Obtain input minimizing function for E

Figure 1: Test case generation scheme.

2 Evolutionary Testing

Evolutionary Testing (ET) refers to software test data generation by means of modern heuristic combinatorial optimization techniques. Typical examples of such techniques are Simulated Annealing (Kirkpatrick et al., 1983), Tabu Search (Glover and Laguna, 1997), and Genetic Algorithms (GAs) (Goldberg, 1989).

The manner in which the heuristic technique takes part remains open, however, the research on the field to date has concentrated basically on structural test data generation through dynamic strategies, whilst the usual heuristic optimization method has been the GA.

Most of the structural ET approaches conform to the same iterative two-step process. In the first step, a previously identified structural entity is selected and marked as an objective. The second step consists of solving the following optimization problem: obtain an input covering the objective entity from the set of possible programme inputs. The objective function is formulated so that if an executed input exercises the objective, the value is minimum, otherwise the value is proportional to the proximity of the input to the objective coverage. In order to obtain the function value of an input, it must be previously executed on an instrumented version of the programme which will provide the necessary information. Figure 1 shows the described general scheme.

Although different strategies exist for selection, the common trend is to determine the objective entity with the help of a graph that reflects the structural characteristics of the programme. For example, in the case of branch coverage, a control flow graph (Fenton, 1985) is usually employed. In this type of graph, a vertex x represents a basic block in the code, i.e. a maximal sequence of statements such that if one is executed then all do. There is an arc (x, y) if the control of the programme can be transfered from block x to y without crossing any other block. Thus, programme branches will come defined by every vertex x with $outdegree(x) > 1$.

The next phase of the scheme in Figure 1 tackles an optimization problem, to be precise, given the search space Ω formed by the programme inputs and a function $h : \Omega \longrightarrow \mathbb{R}$, find $\boldsymbol{x}^* \in \Omega$ such that $h(\boldsymbol{x}^*) \leq h(\boldsymbol{x})\ \forall \boldsymbol{x} \in \Omega$.

Typically, for branch coverage, function h is created according to the following strategy. Given an expression $\mathcal{A}\,\mathbf{OP}\,\mathcal{B}$ of a conditional statement **COND** in the code, with **OP** denoting a comparison

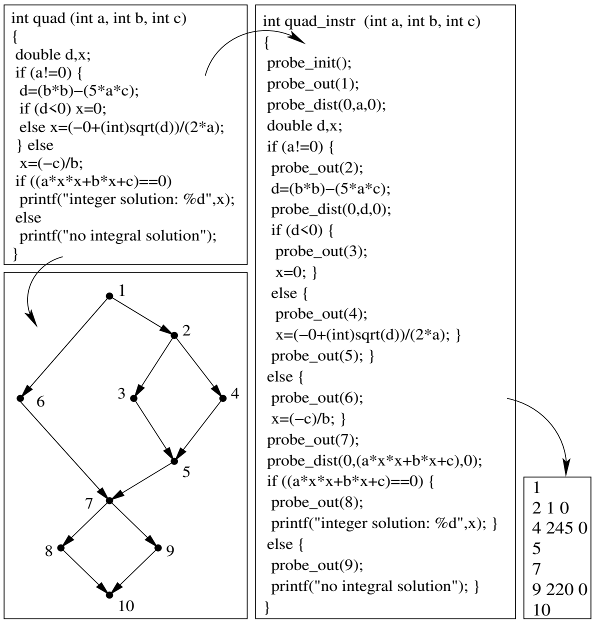

Figure 2: Example function, control flow graph, instrumented version and output file.

operator, the value for an executed input $\boldsymbol{x}$ is determined by the following function:

$$h(\boldsymbol{x}) = \begin{cases} l & \text{if } \textbf{COND} \text{ not reached} \\ d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}}) + K & \text{if } \textbf{COND} \text{ reached and objective not attained} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where l is the highest computable value, $\mathcal{A}_{\boldsymbol{x}}$ and $\mathcal{B}_{\boldsymbol{x}}$ are an appropriate representation of the values taken by $\mathcal{A}$ and $\mathcal{B}$ in the execution, $d(\mathcal{A}_{\boldsymbol{x}}, \mathcal{B}_{\boldsymbol{x}})$ is a distance measurement between $\mathcal{A}_{\boldsymbol{x}}$ and $\mathcal{B}_{\boldsymbol{x}}$, and $K > 0$ is a previously defined constant. Commonly, 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 (Sthamer, 1996).

In the case of 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 h_1 and h_2, and an input $\boldsymbol{x}$, the value for the logical expression $C_1 \vee C_2$ is $min\{h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\}$, the logical expression $C_1 \wedge C_2$ is calculated as $h_1(\boldsymbol{x}) + h_2(\boldsymbol{x})$, and for $\neg C_1$, the value is known by propagating the negation inside C_1. By associating different logical expressions, the overall value for h is obtained.

In order to illustrate how the test case generation can be performed, Figure 2 presents an example function, written in the C programming language, together with its associated control flow graph and its instrumented version. The reduced box on the right represents an output file of a hypothetical execution of the instrumented programme version.

The flow graph is used to select the next objective branch whose coverage will be pursued. Then, an optimization problem is raised, where a solution is, commonly, a representation of the programme input, i.e. three integers in the example.

After an execution of the instrumented code, an output file contains the traversed basic block (numbered as in the control flow graph) at each line and, if the previous block had a conditional statement with an expression $\mathcal{A}\,\mathbf{OP}\,\mathcal{B}$, the values of $\mathcal{A}$ and $\mathcal{B}$. This information is then used to obtain the objective function value of the executed input (solution).

As it can be noticed, this general scheme aims at producing test cases by solving a number of function optimization problems. ET approaches following this idea come motivated by the fact that the space defined by inputs and objective function is usually large and complex. Simple local search methods perform poorly in such spaces as they easily fall into local optima, therefore, more sophisticated search techniques become a suitable alternative.

In (Michael and McGraw, 2001), a GA is applied for condition/decision coverage of C/C++ programmes, that is, every branch needs to be taken and each expression in each conditional statement must be both true and false. In (Jones et al., 1996) and (Sthamer, 1996), branch coverage is sought through a GA, and Simulated Annealing is proposed in (Tracey et al., 1998). The function is smoothed in (Wegener et al., 2001) taking a level of proximity into account for the inputs not reaching the conditional statement of the branch objective, again GAs are used as the search method. In contrast, in the approach presented in (Pargas et al., 1999) the function is just the proximity measurement.

Recently, the described scheme has succesfully been applied for branch coverage making use of EDAs as the optimization method (Sagarna and Lozano, 2003(a)). As the approaches here proposed are based on this work, it is explained in more detail in Section 5.1.1. In (Sagarna and Lozano, 2003(b)), a promising alternative is presented. An EDA searches for the appropriate inputs in a search space which may be modified during the test case generation process.

3 Scatter Search

Scatter Search (SS) (Laguna and Martí, 2003) (Glover et al., 2000) is an evolutionary search method whose principles were introduced in the early sixties. Due to the recent succesful application of SS to several problems, it is actually deserving the attention of the combinatorial optimization community.

SS combines the solutions in a set, named *reference set*, in order to obtain new solutions to be added to the set. One of the main differences between SS and other population based evolutionary algorithms consists in the way selection of the solutions to be combined is performed. In SS, new solutions are created from two or more solutions chosen from the reference set according to a systematic strategy. In contrast, other evolutionary algorithms, such as GAs, obtain a new solution by random selection operators.

Initially, a set P of diverse solutions is constructed. The reference set is formed by extracting adequate solutions from P, i.e. solutions with best objective function value or highest diversity with regard to the solutions already in the reference set. Once the first reference set is obtained, a number of subsets is generated in a systematic way. The solutions of these subsets will be combined to generate new solutions that may replace others in the reference set. To be precise, new solutions are improved through a local search method before considering their inclusion in the reference set. If a new solution is added to the reference set, new subsets are generated and the process is repeated. Otherwise, the algorithm stops.

Structurally, the SS algorithm is composed of five interacting methods. The functionality of each method is clearly specified, however, its definition remains open to the problem being solved. These methods are briefly explained next; for a detailed description refer to (Laguna and Martí, 2003).

Diversification Generation Method

The aim of this method is to generate a number of diverse solutions which will be used to create the set P. The basic SS algorithm stops when no new solution is added to the reference set, nevertheless, a maximum number of iterations is usually established as the stopping criterion. Thus, if no solution is added to the reference set, a rebuilding step is carried out in the next iteration. This rebuiding consists of generating diverse solutions and obtaining a new set P. The reference set is then updated with the solutions in P according to a certain strategy. For instance, replacing half of the solutions in the reference set with worst objective function value, with the solutions in P which most increase the diversity in the reference set.

Improvement Method

Although this method is not strictly required, the common trend is to include it in the SS methodology. Once a solution is obtained, either by diversification or by combination, it is applied an improvement method, typically, a local search method.

Reference Set Update Method

This method manages the reference set by defining the strategies necessary to build and update it. Both, building and updating, can take the objective function value and the diversity of the solutions to be included in the set into account. The building process refers to the solutions in P, while the updating process considers the new obtained solutions for replacement of others already in the reference set, thus maintaining the set size constant.

```
P ← ∅
P ← Add |P| distinct solutions obtained by diversification and improvement
RefSet ← Add the b_1 solutions in P with best objective function value and delete them from P
Repeat for l = 1, 2, ..., MaximumIteration
   RefSet ← Add the b_2 most diverse solutions in P with regard to the solutions in RefSet
   NewElements ← TRUE
   Repeat while NewElements = TRUE
      NewElements ← FALSE
      Generate all new subsets of solutions from RefSet
      Obtain new solutions by combination and improvement
      RefSet ← Update RefSet with new solutions
      If RefSet changed then
         NewElements ← TRUE
      else
         RefSet ← Delete the b_2 solutions with worst objective function value in RefSet
         P ← ∅
         P ← Add |P| distinct solutions obtained by diversification and improvement
```

Figure 3: Pseudocode of basic SS.

Subset Generation Method

The subsets of solutions are generated systematically from the reference set. At least, all subsets formed by two solutions are created. Other implementations consider all subsets of three solutions, four solutions, and so on. As the number of subsets tends to be high, there is a need for keeping a small reference set and, generally, its size is a tenth of the size of P.

Solution Combination Method

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

The interaction of the five methods described above can be observed in the basic SS algorithm proposed in Figure 3. A common size for P is 100 solutions and, therefore, $|RefSet| = 10$. The reference set is usually constructed from the $b_1 = |RefSet|/2$ solutions in P with the best objective function value and the b_2 most diverse solutions. As it can be noticed, the subset generation method only considers the new subsets associated to the solutions introduced in the previous step. If the maximum number of iterations is not reached and no solution has been added to the $RefSet$ then rebuilding is performed, i.e. P is created once again and b_2 of the solutions in the $RefSet$ replaced.

4 Estimation of Distribution Algorithms

The EDA approach (Larrañaga and Lozano, 2002) comprises a set of emerging population based evolutionary algorithms firstly introduced in the field in (Mühlenbein and Paaß, 1996). In contrast to other evolutionary computation heuristics such as GAs, in EDAs, there are neither crossover nor mutation operators. The individuals forming the population of the next generation are obtained sampling a probability distribution previously estimated from the selected individuals. Indeed, this probability distribution is responsible for one of the main characteristics of EDAs, that is, the explicit description of the relationships between the variables defining individuals.

4.1 The abstract EDA

Following concepts refer to discrete domains, that is, variables of individuals are discrete. For a more extended description refer to (Larrañaga and Lozano, 2002) and (Pelikan et al., 1999), in (Larrañaga and Lozano, 2002) continuous domains are also included.

First, some notation which will be used in the rest of the section is required. 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 come 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 will be represented as $p(x_i|x_j) = p(X_i = x_i|X_j = x_j)$. In the context of evolutionary algorithms, an individual of length n is an instantiation $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. The population of the l-th generation will be written D_l and the N selected individuals D_l^{Se} constitute a data set of N cases of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. EDAs estimate $p(\boldsymbol{x})$ from D_l^{Se}, i.e. the probability of an individual being among the selected individuals. This estimation must be carried out at each generation, therefore, the joint probability distribution of the l-th generation will be represented by $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se})$.

A pseudocode for the abstract EDA is presented in Figure 4. The selection step can be carried out using any of the strategies in evolutionary computation, e.g. rank-based selection. Once the individuals are selected, they are treated as a data base from which their probability distribution is estimated. Then this distribution is simulated to create the individuals of a new population. These steps are repeated until a previously defined stopping criterion is met.

The key point of EDAs is how the probability distribution is estimated at each generation. The computation of all the parameters for $p(\boldsymbol{x})$ is unviable, therefore, it is factorized according to a probability model that in some cases, limits the possible dependencies among the variables $X_1, X_2, ..., X_n$.

$Do \leftarrow$ Generate M individuals (the initial population) randomly
Repeat for $l = 1, 2, ...,$until stopping criterion met
$D^{Se}_{l-1} \leftarrow$ Select $N \leq M$ individuals from D_{l-1}
$p_l(\boldsymbol{x}) = p(\boldsymbol{x}\|D^{Se}_{l-1}) \leftarrow$ Estimate the probability distribution
of an individual being among the selected individuals
$D_l \leftarrow$ Sample M individuals (the new population) from $p_l(\boldsymbol{x})$

Figure 4: Pseudocode for abstract EDA.

This leads to several approximations assuming different levels of complexity in their models. The alternatives range from those where the variables are mutually independent, to those with no restrictions to the variables interdependencies. The most restrictive models prevent from the induction of probability distributions having dependencies. However, they allow for a fast and easy estimation that may convert them into a suitable possibility for solving a problem. On the other hand, the least restrictive models are able to show all the dependencies among the variables in a problem although their computational cost is expensive and for some cases, can result in an impractical choice.

Next, a number of EDA approaches are briefly presented. Taking the probabilistic model complexity into account, they are classified as univariate, bivariate and multivariate EDAs. For a more detailed description, including other EDAs, refer to (Larrañaga and Lozano, 2002) and (Pelikan et al., 1999).

4.2 Univariate EDAs

These type of EDAs assume that the n-dimensional joint probability distribution is decomposed as a product of n univariate independent probability distributions, i.e.

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i).$$

UMDA

Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein, 1998) is one of the most simple EDAs. It estimates $p_l(x_i)$ as the relative frequencies of x_i in the data set D^{Se}_{l-1}. That is:

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

where $\delta_j(X_i = x_i|D^{Se}_{l-1}) = \begin{cases} 1 & \text{if } X_i = x_i \text{ in the } j\text{-th case of } D^{Se}_{l-1} \\ 0 & \text{otherwise.} \end{cases}$

4.3 Bivariate EDAs

These approximations make use of second order statistics in order to estimate the joint probability distribution. Therefore, apart from the probability values, a structure that reflects the dependencies must be learnt. The factorization carried out by the models from this category can be expressed as follows:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i|x_{j(i)})$$

where $X_{j(i)}$ is the variable, if any, on which X_i depends.

TREE

TREE (Larrañaga and Lozano, 2002) refers to an adaptation of the Combining Optimizers with Mutual Information Trees (COMIT) algorithm (Baluja and Davies, 1997). The COMIT algorithm hybridizes the EDA approach with local search methods, while TREE only involves the EDA part. The estimation of the probability distribution is carried out using the Maximum Weight Spanning Tree (MWST) algorithm (Chow and Liu, 1968), which is based on the construction of a tree structure. Initially, the weight of each candidate branch (X_i, X_j) is calculated as the mutual information measurement between X_i and X_j. Then, branches with the highest weight are increasingly added in the tree if they do not form a cycle in the graph. Once the structure is terminated, $p_l(\boldsymbol{x})$ is obtained through the conditional relative frequencies in D_{l-1}^{Se}.

Notice that in this model, two distinct variables may depend on the same.

4.4 Multivariate EDAs

With the models from this category, the expression of all the existing variable interdependencies in a problem is possible. The probability distribution is estimated by means of probabilistic graphical models (Castillo et al., 1997), which use a graph to represent the detected interdependencies between the variables. Thus, the factorization associated to this type of EDAs is as follows:

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

where $\boldsymbol{pa_i}$ are the instantiations of $\boldsymbol{Pa_i}$, the set of variables on which X_i depends.

EBNA

In the Estimation of Bayesian Network Algorithm (EBNA) (Larrañaga et al., 2000), the factorization of the joint probability distribution is given by a Bayesian network (BN) learnt from D_{l-1}^{Se}. A BN is a pair $(S, \boldsymbol{\theta})$ where S is a directed acyclic graph representing the (in)dependencies among the variables and $\boldsymbol{\theta}$ is the set of conditional probability values needed to define the joint probability distribution.

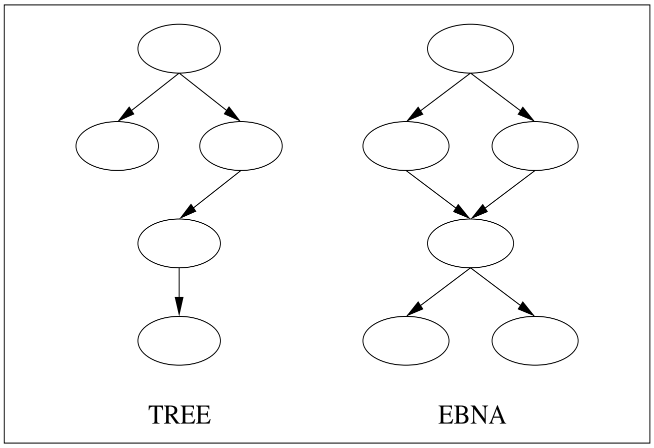

Figure 5: Example of structures for EDAs.

The method used to learn the structure S leads to different EBNA instantiations, for example, $\mathrm{EBNA}_{\mathrm{K2+pen}}$ seeks for the structure with the best penalized K2 (Cooper and Herskovits, 1992) score metric.

Regardless of how the BN is learnt, EBNA simulates $p_l(\boldsymbol{x})$ by means of Probabilistic Logic Sampling (Henrion, 1988).

In Figure 5 an example of the structures associated to the TREE and EBNA algorithms are depicted.

5 Performance Evaluation of pure Scatter Search approaches

The ET based test case generator here proposed deals with branch coverage for C/C++ programmes. This generator is based on the work described in (Sagarna and Lozano, 2003(a)) but, instead of using an EDA as the optimization method, several SS alternatives are considered here, leading to different approaches. The performance of these approaches is evaluated according to a number of criteria and compared to the previous EDA based system. Next, the general framework of the generator and the common characteristics of all the SS approaches are explained.

5.1 General framework

The difference between the work in (Sagarna and Lozano, 2003(a)) and the approaches here presented consist of the applied search method in the optimization phase. Here, all the alternatives make use of an SS algorithm with a number of common features for all the approaches. As the test case generator's framework is the same as in the referred work, this last is briefly described next.

$\boldsymbol{x} \leftarrow$ Translate individual $\boldsymbol{i}$ to input

Execute instrumented programme with $\boldsymbol{x}$

Repeat for each uncovered branch b

    $f_b^{\boldsymbol{i}} \leftarrow h_b(\boldsymbol{x})$

    $f_b^{\boldsymbol{w}} \leftarrow$ Find the fitness of the worst individual

    $\boldsymbol{w}$ in the set associated to b

    If $f_b^{\boldsymbol{i}} < f_b^{\boldsymbol{x}}$

        Substitute $\boldsymbol{w}$ by $\boldsymbol{i}$ in the set associated to b

        If $f_b^{\boldsymbol{i}} = 0$

            Mark b as covered

        else

            Mark b as untreated

    If b is the objective

        $fitness \leftarrow f_b^{\boldsymbol{i}}$

Return $fitness$

Figure 6: Pseudocode for the evaluation algorithm.

5.1.1 Test case generator framework

The system exposed in (Sagarna and Lozano, 2003(a)) conforms to the general scheme in Figure 1. Each code branch is associated with three possible states, covered, treated but uncovered or untreated. The stopping criterion of the scheme is full coverage achievement or unsuccessful treatment of every unexercised objective branch.

The optimization phase is tackled by means of an EDA. Each individual is a 0-1 string representing an input, and its fitness is the value taken by function h (equation 1). Implementation of h is based on the function type also described in Section 2.

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 branch. If the branch is exercised, its state is marked as covered and the input is stored. Otherwise if the fitness of the individual is better than the worst individual in the set associated to the branch, then this last is replaced in the set by the new better individual and, if the branch had previously been treated, its state is marked as untreated. Figure 6 shows the evaluation algorithm for an individual $\boldsymbol{i}$, the value of function h associated to branch b for a given input $\boldsymbol{x} \in \Omega$ is represented by $h_b(\boldsymbol{x})$.

The EDA finishes when either the minimum is found, i.e. an input covering the objective branch

$\overline{f}_{best} \leftarrow \infty$

$objective \leftarrow \emptyset$

$tie \leftarrow false$

Repeat for each untreated branch b

  $\overline{f}_b \leftarrow$ Average fitness of the individuals associated to b

  If $\overline{f}_b < \overline{f}_{best}$

    $\overline{f}_{best} \leftarrow \overline{f}_b$

    $objective \leftarrow b$

  If $\overline{f}_b = \overline{f}_{best}$

    $tie \leftarrow true$

If $tie = true$

  $objective \leftarrow$ Breadth first search between branches with $\overline{f}_{best}$ value

Return $objective$

Figure 7: Pseudocode for the selection algorithm.

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.

Apropos the objective branch selection phase, a control flow graph is used to identify the branches at the initialization stage and help to decide the branch to be selected next during the process.

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 best. In case a tie is given, then a breadth first search is carried out, i.e. from the tied branches, selection of the one with the lowest level in the control flow graph. A pseudocode of the selection algorithm can be observed in Figure 7.

The 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 mean fitness in the set is better than before optimization and could result in a promising population seed.

5.1.2 Features of SS

As in the previous work, all the SS approaches codify a solution as a 0-1 string representing an input and, obviously, the objective funtion value is given by the value of h for the represented input. The same evaluation scheme as in the referred work is applied during the SS execution, that is, each solution is evaluated according to every non-objective branch.

In order to include the seeding strategy described in Section 5.1.1, the set of inputs of an objective branch, which has the same size as set P, is overturned in P at the begining of the SS. Thus, in practice, the set of inputs can be viewed as a particular initial P set for each branch. At the test case generation process startup, the set of inputs of each branch b is created by introducing distinct solutions obtained via diversification and, if it is the case, improvement. The evaluation of a solution is not only performed with regard 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 already in the set, 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 with the same seed. When starting the creation of the set of inputs for b, a number of solutions may already be in it, however, they are not improved with regard to b because they were found during the improvement of other branch, so the improvement method is applied to these solutions. Nevertheless, whatever the number of solutions 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.

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 the reference set, a rebuilding step is carried out.

5.2 Pure SS methods

In order to evaluate the performance of the SS in solving the test case generation problem, several approaches were developed. These SS approaches come defined by the performed implementations of the interacting methods described in Section 3. These implementations are explained as following.

Diversification Generation Method

This method can create a previously fixed number of diverse solutions in the following two ways:

all random Each solution is randomly generated according to a uniform distribution.

random/complementary Half of the solutions are created through a uniform distribution, while the other half are the complementary solutions of the randomly created.

The rebuilding step consists of replacing half of the solutions in the reference set with the worst values of h, with the solutions in P which increase the most the diversity in the set. In this step, the generation of set P comprises the next three possibilities:

input set Overturning the set of inputs associated to the objective branch into P.

input set/diverse Filling half of P with diverse (and improved) solutions and overturning the other half with the inputs in the set of inputs which most increase the diversity in P.

all diverse Obtaining all the solutions via diversification (and improvement).

Improvement Method

Apropos the improvement method, the following options were implemented:

no improvement The SS approach includes no improvement method.

exhaustive local search An exhaustive local search where the neighbourhood is defined by the solutions with a Hamming distance of one to the current solution.

best first local search A local search with the same neighbourhood as the previous, but at each step the next solution is the first that improves the current. If the current solution $\boldsymbol{x}$ differed in the variable i from the previous, the neighbours of $\boldsymbol{x}$ are examined starting from variable $i+1$ in steps at one at a time. Obviously, if the last variable is reached, the first is next considered. The search stops when no improved solution is found after a complete turn.

restrictive parameter walk In this local search, neighbours also are to a Hamming distance of one from the current solution, however, the bit substrings codifying each input parameter are taken into account to create a neighbour solution. For a solution differing from the previous in the i-th variable of the substring codifying the j-th input parameter, there is a unique neighbour, which is obtained by changing the value of the most significant bit, previously unchanged, in the substring associated to the next input parameter. To be precise, 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.

best first parameter walk This method is similar to the best first local search, however, in this case, the order in which the variables are examined is different. Neighbour solutions are considered in the order given by the strategy in the restrictive parameter walk and the search stops when no better solution is found after a complete turn.

As an illustration, Figure 8 presents different examples of the local search methods above described. The solution at the top is the initial solution codifying an input with three parameters; a vertical line separates the substrings representing each parameter. It is supposed that this initial solution was

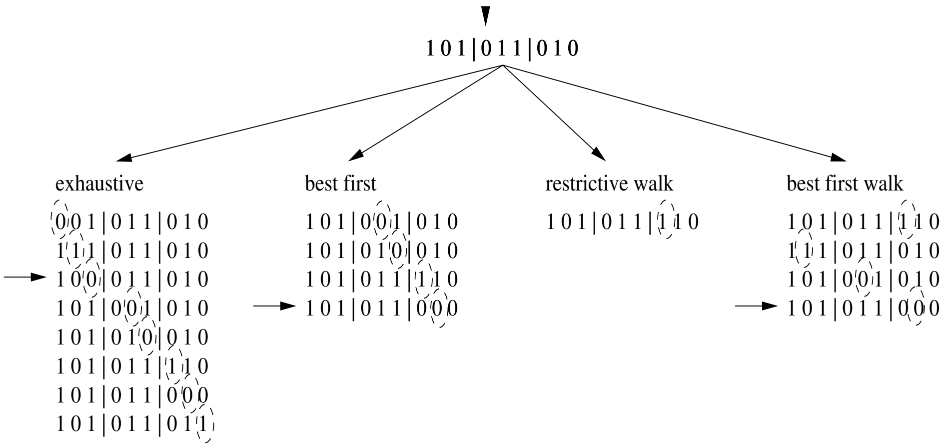

Figure 8: Example of local seach improvement methods.

obtained by changing the signed bit. For each local search method, the neighbours considered in a hypothetical search step are shown. The horizontal arrow indicates the assumed new best solution chosen for the next step. Restrictive parameter walk has no arrow since the neighbour did not improve the initial solution. As could be noticed, the parameter walk based methods take the particular characteristics of the problem into account, as they give a higher priority in the search to variables which can lead to a significant change in the input parameter values.

Reference Set Update Method

Both the reference set building and updating implemented alternatives conform to the strategies described in (Laguna and Martí, 2003).

Given a reference set of size b, it is built by first choosing the $b/2$ solutions from P with the best objective function value, and then increasingly introducing the $b/2$ solutions with maximum diversity with regard to those already included. The diversity of a solution $\boldsymbol{x}$ is measured as $\min\{d(\boldsymbol{x}, \boldsymbol{x'}) | \boldsymbol{x'}$ is a solution in the current reference set$\}$, where $d(\boldsymbol{x}, \boldsymbol{x'}) = \sum_i |x_i - x'_i|$.

The implementation of the reference set updating follows a *static update* strategy. New solutions obtained by combination (and improvement) are placed in a pool. Once the pool is full, the reference set is constructed with the solutions with the best h value from the current reference set and the pool.

Subset Generation Method

The subsets of solutions used for combination consist simply of all the pairs of solutions in the reference set. Obviously, only the pairs of solutions not previously generated are taken into account, i.e. pairs of new solutions are considered.

Solution Combination Method

From each subset of solutions, one new solution is obtained according to the probabilistic rule presented in (Laguna and Martí, 2003).

For each variable i, a score is calculated based on the objective function value of the solutions $\boldsymbol{x}$ and $\boldsymbol{x'}$ in the subset, that is

$$score(i) = \frac{h(\boldsymbol{x})x_i + h(\boldsymbol{x'})x'_i}{h(\boldsymbol{x}) + h(\boldsymbol{x'})}$$

The new solution is constructed by using this score as the probability of variable i taking the value 1.

5.3 Pure SS experimental results

In order to observe how the presented approaches perform in practice, several experiments were carried out. Pure SS approaches are evaluated and compared with regard to the results in (Sagarna and Lozano, 2003(a)), taking all the programmes tested in this last work into account. These programmes are the following:

Triangle This is a popular programme 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 to the input. Four different versions were used. The Triangle1 programme (Wegener et al., 2001) owns three integer parameters which, in the experiments, took values in the interval $[-16384, 16383]$; the number of branches (objectives) to be covered is 26. Triangle2 (Wegener et al., 2001) is the same as Triangle1 with floating point parameters instead, the interval for each was $[-32768, 32768]$. On the other hand, Triangle3 (Michael and McGraw, 2001) is a new implementation with integer parameters for which the interval $[-512, 511]$ was chosen; the number of branches is 20. Finally, Triangle4 (Sthamer, 1996) constitutes a distinct implementation once again, the selected interval for its integer parameters was $[-512, 511]$; 26 branches are included.

Atof Given a string of characters as input, Atof (Wegener et al., 2001) 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). The number of branches is 30.

Remainder This function (Sthamer, 1996) calculates the remainder of the division of two integer numbers, therefore an input is composed of two interger parameters for which the interval $[-32768, 32767]$ was chosen during experimentation. The Remainder programme has 18 branches.

Complexbranch In this case, there is no specific functionality as it is a function artificially created for testing purposes (Wegener et al., 2001). Its main characteristic is the existence of several

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	3369.5 68.46*	6299.5 94.62*	4435 96.92	3623.7 63.08*	3157.5 96.15	4292.3 96.15*
exhaust. search	16743.4 100	21563 100	19607.2 100	19401 100	15843 100	15289.2 100
best first search	8859.7 100	11353.5 100	12398.3 100	8008.4 100	8938.6 100	9463.1 100
restrict. walk	6654.4 74.62	5183.1 96.92	7219.1 96.92	7056.6 73.85*	6573.4 96.54	6808.4 97.69
best first walk	10117.1 100	**7957.9** **100**	11176.8 100	8917 100	8324.8 100	11036.6 100

Table 1: Results for Triangle1 with all pure SS approaches.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	5092.3 46.54*	9432.4 83.85*	13905.7 93.08*	5477.9 42.31*	8820.3 88.08*	16710.7 93.46*
exhaust. search	36256.5 89.62	18512.2 100	22343.9 100	63433.1 89.23	26248.5 99.62	20198.1 100
best first search	18121.5 96.15	30328.9 99.62	23356.5 99.62	28118.9 96.54	24293.4 98.85	22577.7 100
restrict. walk	12227.5 50.77*	15127.9 95.38	18313.6 95.38*	8812.5 63.85*	16670.7 93.85*	12839.1 95.38*
best first walk	17118 98.85	21371.5 99.23	**14195** **100**	42271.8 95.38	20088.3 98.85	17311.8 100

Table 2: Results for Triangle2 with all pure SS approaches.

hard to cover branches in the code. Six integer parameters form an input taking values in the interval $[-512, 511]$; the source code reveals 22 branches.

All the SS approaches which can be obtained from the combination of the different implemented methods was considered with regard to each programme. Then, the best approaches from these experiments were taken for comparison with the best results obtained in (Sagarna and Lozano, 2003(a)).

The maximum number of generations for all the SS alternatives was set to 10, the size of set P was 100 and the reference set size was 10. The results of the experiments for all the programmes are shown in Tables 1 to 7. In each cell, the average results from 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. The values presented in bold correspond to the SS with highest coverage and, in case a tie is given, lowest number of generated inputs (solutions) during the whole process.

In order to clarify the results of Tables 1 to 7, SS approaches were ranked by following the algorithm in Figure 9 for each programme. The confidence level for the tests was set to 0.05. For the ordering of the approaches, two criteria were considered:

- On the one hand, complete branch coverage is adopted as a mandatory requirement in some

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	1812.5	2801.1	3978.6	1534.7	2320.9	3514.6
	82*	87.5	91.5*	85.5*	91.5*	91.5*
exhaust. search	42743.3	45941.6	55344.4	60067.5	41802.1	52307
	91.5	95.5	95.5	95	97	96
best first search	63678.8	42930.7	44524.9	63480.2	**39966.1**	55140.5
	94.5	96	97	91.5	**98**	96
restrict. walk	4480.2	6608.1	6308.3	4685.4	5129.6	7706.2
	88	90.5	95	86.5*	95	96.5
best first walk	52106.3	41751.1	56587.9	54925.2	44878.8	46235.7
	92.5	97.5	97.5	89.5	96.5	97

Table 3: Results for Triangle3 with all pure SS approaches.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	5050.3	20375.3	39738.4	5292	21378.6	39009.6
	79.62*	89.23*	90.77*	81.92*	89.62*	89.62*
exhaust. search	58753.2	**48216.6**	58540	63024	48795	57555.6
	100	**100**	98.85	99.23	99.23	99.23
best first search	34894.7	48460.1	39374.7	37641.1	31782.3	35926.3
	98.46	98.08	99.23	98.46	98.85	99.62
restrict. walk	13692.2	53076.9	72941.5	14897.6	42798.3	55531.3
	89.23*	88.46*	90.77*	88.46*	89.62*	90*
best first walk	25852.9	30783.8	42402	26857.5	34121.8	36803.3
	99.23	97.69	98.46	98.85	98.08	97.69

Table 4: Results for Triangle4 with all pure SS approaches.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	6069.8	15395.4	20947.6	6118.9	13510.2	18948.3
	48.67*	64*	64.67*	46.33*	64*	64*
exhaust. search	442720.9	473041.4	628005	407709.4	545354.2	689438.3
	62.67*	88.67	90.67	66*	84.33	94
best first search	458468.9	587434.2	514446	469608.2	525194.6	781493.5
	61.33*	79.33	93.33	60.67*	85.67	87.33
restrict. walk	13107.1	27577	44717.8	13367.3	26966.3	44753.7
	53.33*	65.33*	66.67*	52.33*	64.67*	66.67*
best first walk	435082.8	538065.6	**596952.5**	474762.2	562762.9	630403.9
	64*	83.33	**94**	60*	78.67	88.67

Table 5: Results for Atof with all pure SS approaches.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	229.7	224.8	346.2	145.9	159.1	123
	100	100	99.44	100	100	100
exhaust. search	1109.7	1514.1	921	830	697	802.2
	100	100	100	100	100	100
best first search	245.7	241.7	231.1	154.4	180.2	138.6
	100	100	100	100	100	100
restrict. walk	600	634.8	705.9	537.3	566.1	505.6
	100	100	100	100	100	100
best first walk	**107.3**	123.2	164.2	120.2	109.5	133.9
	100	100	100	100	100	100

Table 6: Results for Remainder with all pure SS approaches.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	2608.5	3688.4	3844.8	2624.5	3559.1	4238.2
	85.91*	93.18*	94.09*	85.45*	90.91*	93.18*
exhaust. search	28606.2	45426.2	29854.5	27692.6	33298.6	**26928.9**
	97.73	98.18	100	97.73	99.55	**100**
best first search	37679.5	33086.4	30526.9	28175.6	28992.9	33971.2
	95.91	99.55	100	96.82	99.55	100
restrict. walk	6948.4	6672.1	6705.5	6500	7712.1	7350.3
	86.82*	94.55	95	90*	94.55*	93.64
best first walk	25623.7	58896.7	27879.4	28749.6	40510.1	37058.9
	97.73	97.27	100	97.27	99.09	99.55

Table 7: Results for Complexbranch with all pure SS approaches.

quality standards. Thus, SS approaches were ordered by their coverage measurement (non-increasing order); if a tie was given, involved approaches were ordered by their variance (non-decreasing order).

- On the other hand, it is important for a test case generator to obtain a coverage value with the lowest cost, that is, producing as less inputs as possible. Therefore, the number of generated inputs was used as the criterion for another ordering (non-decreasing order); in case of tie, involved approaches were ordered by their variance. Care must be taken with this criterion, since for a low number of generated inputs the coverage may be poor, e.g. the lowest number of inputs for the Atof programme is 6069.8 and corresponds to a SS with 48.67% coverage. Thus, only the set of approaches reaching an appropriate coverage value were considered in the ordering. To obtain these approaches, a *frontier* value was stablished at a 10% less than the highest achieved coverage. Then, the included approaches in the set were those with a larger or equal coverage than the frontier and, from the others, those with no statistically significative different coverage than any of the closest approaches to the frontier. The SS alternatives not included in the ordering were assigned the worst value in the rank, that is, 30.

In Table 8, the sum of the ranks of each SS for all the programmes is presented. The first row in each cell corresponds to the coverage criterion and the second belongs to the inputs criterion. Best values are marked in bold.

Comparing both halves of Table 8, values are similar, thus suggesting there are no differences between the diversification methods. In fact, the Mann-Whitney test, applied to each criterion, revealed no significative differences between both methods at a confidence level of 0.05.

Apropos rebuilding, best rank values in the case of coverage criterion are for "all diverse" rebuilding approaches, followed by "input set/diverse" and "input set" alternatives respectively. According to these results, coverage decreases as the leading role of the set of inputs increases (Tables 1 to 7 show this for most of the programmes). A possible explanation for this may be the existence of a lack of diversity in

$SS_{1:n}, SS_{2:n}, ..., SS_{n:n}$ $\leftarrow$ Order SS approaches according to some criterion

$\pi(1) = 1$

m $\leftarrow$ Cases of $SS_{1:n}$

m' $\leftarrow$ Cases of $SS_{2:n}$

Repeat for $i = 2, ..., n$

  Perform a Mann-Whitney test over m and m'

  If significative differences

    $\pi(i) = \pi(i-1) + 1$

    $m \leftarrow m'$

  else

    $\pi(i) = \pi(i-1)$

    m $\leftarrow$ Add m' to m

  m' $\leftarrow$ Cases of $SS_{i+1:n}$

return π

Figure 9: Ranking algorithm.

improv. m.	all rand. div.			rand./compl. div.		
	input set reb.	input set/div. reb.	all div. reb.	input set reb.	input set/div. reb.	all div. reb.
no improv.	28	21	16	29	20	17
	154	68	42	152	68	40
exhaust. search	15	8	**7**	13	**7**	**7**
	81	25	24	82	24	23
best first search	12	**7**	**7**	12	8	**7**
	47	20	20	46	17	17
restrict. walk	26	16	14	26	15	14
	129	73	49	129	46	47
best first walk	11	8	**7**	13	8	**7**
	44	**16**	17	46	**16**	**16**

Table 8: Rank of pure SS approaches according to coverage and inputs criteria.

Programme	SS_{best}	$\text{EBNA}_{\text{K2+pen}}$	TREE
Triangle1	11036.6	6120	2945
	100	99.62	98.46
Triangle2	17311.8	9360	4420
	100	100	98.46
Triangle3	46235.7	6660	3700
	97	100	96
Triangle4	36803.3	22720	10535
	97.69	100	89.23*
Atof	630403.9	9505	14350
	88.67	100	87.33
Remainder	133.9	2360	2360
	100	100	96.11
Complexbranch	37058.9	11930	6885
	99.55	100	94.55

Table 9: Results of best SS and EDA approaches.

the set of inputs of the objective branch. In general, the previous order of ranks also holds in the inputs criterion, however the difference between "input set" rebuilding and the other two methods is higher than in the coverage criterion. The reason for this is that there exist more approaches not reaching an acceptable coverage value (and, thus, penalized with a rank value of 30) for the input set alternative than for the others.

Assuming the "exhaustive" local search, "best first" search and "best first parameter walk" perform a more intensive search than the other improvement methods, best rank values are for the most intensive. Therefore, the improvement method seems to be an important element of the SS in order to obtain high quality solutions.

Table 8 shows that the alternative with "random/complementary" diversification, "all diverse" rebuilding and "best first parameter walk" takes the best rank values for the two criteria. Thus, according to these results, it can be concluded that this is the best overall SS approach.

The best SS approach was compared with the results in (Sagarna and Lozano, 2003(a)). In this last work, for each EDA different population sizes and several values for the maximum number of generations were considered. The set of EDAs used in the experiments comprised all the introduced in Section 4 and three more: Population Based Incremental Learning (Baluja, 1994), Mutual Information Maximization for Input Clustering (De Bonet et al., 1997) and EBNA_{BIC} (Larrañaga et al., 2000). EDAs were ranked in a similar manner to the previous in order to identify the best overall. $\text{EBNA}_{\text{K2+pen}}$ and TREE were the best for the coverage and inputs criteria respectively. Thus, for each programme, the best configuration of TREE and $\text{EBNA}_{\text{K2+pen}}$ for the criteria were compared with the best SS. Table 9 presents the results. SS_{best} denotes the previously identified best SS; the format of each cell is the same as in Tables 1 to 7.

Table 9 shows that, in general, EDAs outperform SS approaches. In Triangle1 and Remainder programmes, coverage results of SS_{best} are better than those of EDAs. However, for the rest of the programmes, coverage of SS_{best} is either lower or equal but with more inputs than $\mathrm{EBNA}_{\mathrm{K2+pen}}$. The number of generated inputs in Remainder is lower than in the EDAs; unfortunately, this does not hold for the other programmes, although the coverage in SS_{best} is higher than in TREE.

6 SS-EDA hybrid approaches

In (Sagarna and Lozano, 2003(a)), EDAs were applied with great success, clearly outperforming the results in previous works (Sthamer, 1996),(Michael and McGraw, 2001),(Wegener et al., 2001). On the other hand, according to the experimental results from Section 5.3, SS approaches reaching upper coverage values generate, in general, a higher number of inputs. These observations motivated the idea of combining both optimization methods. As a result, several hybrid alternatives were obtained and their performance measured by using the programmes described in Section 5.3.

6.1 Hybrid approaches

Developed SS-EDA hybrid approaches consist of a SS algorithm, with no improvement method, which uses an EDA before the construction of set P. The EDA operates in the same manner as described in Section 5.1.1. Two kind of hybrids can be distinguished: SS with EDA in initial P and SS with EDA in initial P and rebuilding.

As it was explained in Section 5.1.2, the initial P set is obtained by overturning into it the set of inputs associated to the objective branch. Thus, the former type of hybrids apply an EDA before this overturning. To be exact, the set of inputs is taken as the initial population of the EDA and, when it finishes, the resulting set of inputs is used to create P.

In the other kind of hybrids, the initial set P is identically created as in the former hybrids. However, in this case the same P construction strategy is used in the rebuilding step. Notice that, if the selected option for the rebuilding is the "all diverse" strategy, then the effect of the EDA will not be inmediately reflected on P.

6.2 Performance of hybrid approaches

Hybrid approaches were applied to the previous programmes in order to observe their performance. In the experiments, the best pure SS alternative (SS_{best}) was taken as the basic SS algorithm to configure an hybrid. As no improvement method is used in the hybrids, only the diversification and rebuilding of SS_{best} can be used in each configuration. Thus, two distinct types of approaches are considered: SS_{best} based and initial P hybrids (denoted as SS^{i}_{best}) and SS_{best} based and initial P and rebuild hybrids (denoted as SS^{ir}_{best}).

Programme	SS^{i}_{best}-EBNA$_{\mathrm{K2+pen}}$	SS^{ir}_{best}-EBNA$_{\mathrm{K2+pen}}$	SS^{i}_{best}-TREE	SS^{ir}_{best}-TREE
Triangle1	3294	7178.7	3473.8	4040.8
	96.54	97.69	96.92	99.23
Triangle2	5418.5	11310.8	5453.8	6054.5
	96.15	98.46	97.31	98.85
Triangle3	3073.5	5955	3115.4	6373.3
	96	98.5	96.5	99
Triangle4	14240.4	30502.2	26875.2	18880.6
	91.15	91.92*	95.38	95.38
Atof	25586.6	17140.9	23252.7	46903.4
	70*	99	76.33*	96
Remainder	811.9	689.3	841.7	994.2
	100	100	100	100
Complexbranch	4445.7	7663.7	4215.6	8336.6
	96.82	99.09	96.82	98.18

Table 10: Results of SS-EDA hybrid approaches.

A hybrid approach is completely defined once the EDA is specified. For the experiments, the best EDAs in (Sagarna and Lozano, 2003(a)) were selected, i.e. TREE and EBNA$_{\mathrm{K2+pen}}$. 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 (Henrion, 1988), and the population was created in an elitist way. Population size was the same as the one for set P and the maximum number of generations was set to 10. Table 10 shows the results of the experiments; once again, the cell format is the same as in previous tables.

The Mann-Whitney test revealed no statistically significative differences ($\alpha = 0.05$) between EBNA$_{\mathrm{K2+pen}}$ and TREE based hybrids. This means that there is no difference in the results whatever the chosen EDA.

It can be observed that initial P based hybrids obtain less inputs and a lower coverage than the initial P and rebuild alternatives. This is an intuitive result, since the former type of hybrids only use an EDA at the initial stage and act as a SS with no improvement for the rest of the search. However, an hypothesis test showed no differences ($\alpha = 0.05$) in the number of inputs but statistically significative differences in the coverage. Thus, it can be concluded that initial P and rebuilding hybrids are better than initial P for the coverage criterion.

In the bulk of the cases, the coverage results in Table 10 do not improve those in Table 9. In this last table, the best coverage for the EDAs is better or equal than for the hybrids. With regard to SS_{best}, the coverage is outperformed by the hybrids in Triangle3 and Atof, and is equal in the Remainder programme.

The best number of inputs generated by the hybrids is lower than for the EDAs in half of the programmes. SS_{best} generates more inputs than the best hybrids in all the programmes, with the exception of Remainder. The Mann-Whitney test revealed no differences ($\alpha = 0.05$) in the number of inputs among the hybrids and, therefore, it can be concluded that the hybrid approaches generate, in general, less inputs than SS_{best}.

7 SS-EDA cooperative approaches

All the previous approaches, either SS or EDAs based, aim at generating test data for a given programme by their own. However, these approaches could be entirely used in order to deal with the same problem, thus leading to a cooperative approach which profits from the benefits of SS and EDAs.

7.1 Cooperative approaches

In these approaches, SS and EDAs act separately with the aim of solving the same test case generation problem. Therefore, a cooperative approach consists of using an EDA according to the work explained in Section 5.1.1 and, once it has finished, applying one of the pure SS approaches described in Section 5.2 over the remaining uncovered objectives. This way, the general scheme in Figure 1 is first applied with an EDA and, if it was not able to solve the complete problem, the scheme is repeated with SS for the uncovered branches.

Two alternatives are proposed for the cooperation. On the one hand, the SS approach initializes the set of inputs of each uncovered branch via the diversification method. On the other, the set of inputs is borrowed from the resulting set after the EDA based approach.

7.2 Results of cooperative approaches

Cooperative approaches were applied to the programmes described in Section 5.3 in order to observe their performance. Again, SS_{best} was taken in the experiments as the basic SS algorithm for the cooperations.

SS_{best} was used in the two types of cooperative approaches previously described, leading to two kinds of approaches: SS_{best} with initialization via diversification method (C^{d}_{best}) and SS_{best} with initialization via the resulting set of inputs from the EDA (C^{r}_{best}).

It should be noted that, in C^{r}_{best}, the diversification method will only be used for the rebuilding step, while, in the other kind of approaches, SS does not benefit from the resulting set of inputs after the EDA.

As ocurred in the hybrids, a cooperative approach is completely defined when the EDA is specified. The same EDAs and parameters as in the hybrid approaches were used for the experiments. The results are presented in Table 11, where each cell conforms to the format of previous tables.

The Mann-Whitney test found no statistically significative difference ($\alpha = 0.05$) between $\mathrm{EBNA}_{\mathrm{K2+pen}}$ and TREE based approaches. Therefore, there is no difference in the effect of the

Programme	C^d_{best}-EBNA$_{K2+pen}$	C^r_{best}-EBNA$_{K2+pen}$	C^d_{best}-TREE	C^r_{best}-TREE
Triangle1	7493.2	8373.8	8017.1	2658.4
	100	100	100	100
Triangle2	11910.3	4728.7	22095.3	7265.3
	99.62	100	98.85	99.62
Triangle3	35161.1	9026	26488.4	11728
	96.5	99.5	98	99
Triangle4	32331.3	29863	31730.2	21672.8
	98.85	98.08	99.23	98.46
Atof	728371.9	185933.9	806890.9	169371.3
	84.67	96.67	82.33*	96.67
Remainder	2081.3	3868.8	1816	3727.6
	100	100	100	100
Complexbranch	31420.5	29457	30347	40158.2
	100	100	100	100

Table 11: Results of cooperative approaches.

chosen EDA over the results.

With statistically significative differences ($\alpha = 0.05$), C^r_{best} alternatives offer, in general, better results than C^d_{best} based. In the bulk of the programmes, the coverage value in the formers is equal or higher than in the laters, while the number of generated inputs in these is larger than in C^r_{best} approaches. This suggests that using the search results obtained from the previous optimization method may be useful for the actual search.

Figures 10 and 11 present, respectively, the proportion of coverage and generated inputs by the EDA based and SS based optimization methods. As it can be noted in Figure 10, the EDA based method 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 its own, whilst this can be obtained by using the SS method. In fact, the SS based method always augments the coverage attained by the EDA based. On the other hand, Figure 11 shows that, in most of the cases, the proportion of generated inputs is higher in C^d_{best} than in C^r_{best} approaches. This is an intuitive result since, in C^d_{best} alternatives the SS method starts from a set solutions generated via diversification (and improvement). By contrast, C^r_{best} approaches seed the SS method with the best solutions found by the EDA. Although this may involve a lose of diversity, the SS is supposed to recover from it through the rebuilding step.

Generally, the coverage of cooperative approaches is equal or outperforms the attained by SS_{best} and the best hybrid. In half of the programmes, the best value in Table 11 is equal or better than the coverage of the best EDA.

With regard to the number of inputs, the best cooperative approach outperforms SS_{best} in all the programmes unless Remainder. However, in most of the cases, this does not hold for the best EDA and hybrid, and the value of the best cooperative approach is worse.

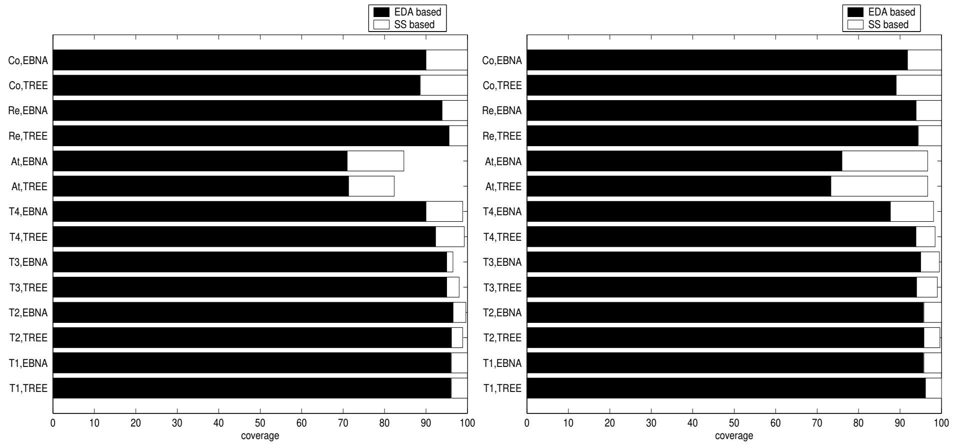

Figure 10: Proportion of the coverage in C^d_{best} (left) and C^r_{best} (right).

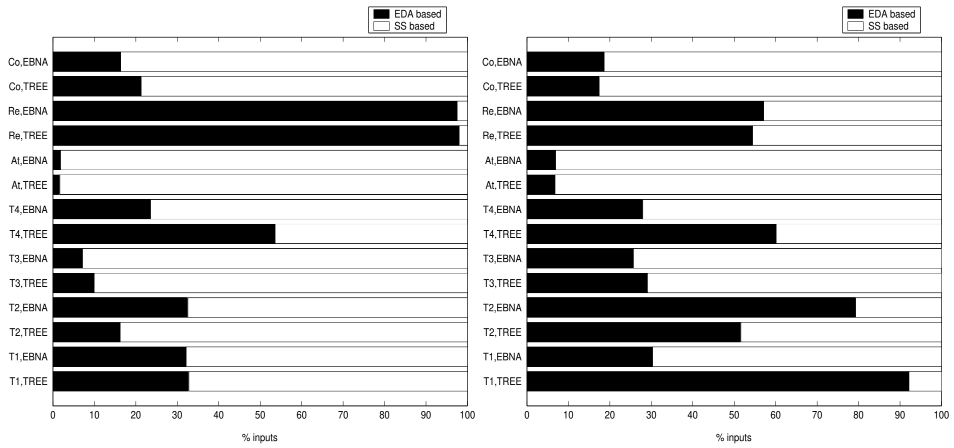

Figure 11: Proportion of the generated inputs in C^d_{best} (left) and C^r_{best} (right).

8 Conclusions

In this work, a new alternative for the automatic generation of test cases was described. In order to fulfil the branch coverage criterion, test inputs are searched by means of several SS based approaches. These are grouped in pure SS, SS-EDA hybrids and cooperative approaches. To the best of our knowledge, SS had never been previously used to solve this problem, and it is the first time SS and EDAs are combined.

The obtained results from the experiments, suggest that the improvement method plays a main role in the search for high quality solutions. It can be concluded that more advanced SS strategies should be studied, as well as thoroughly exploring all the possible parameter values.

The experiments conducted over the SS and EDAs combinations offered encouraging results. SS-EDA hybrids improved the results of the pure SS approaches according to the number of generated inputs. Cooperative alternatives improved results in other works and those of the pure SS and SS-EDA hybrid approaches as well. The use of SS as a secondary otimization method improved the coverage of the previous EDA based method, thus showing useful the cooperation strategy. These cooperative scheme presents as a powerful option for tackling this and other problems and, therefore, it is a promising line of work for further research.

Acknowledgements

This work was supported by the Ministry of Science and Technology under grant TIC2001-2973-C05-03. R. Sagarna was also supported by the Department of Education, Universities and Research of the Basque Government under the programme of Formation of Researchers.

References

Baluja, S. 1994. Population Based Incremental Learning: A method for Integrating Genetic Search Based Function Optimization and Competitive Learning. *Technical Report CMU-CS-94-163.* Carnegie Mellon University, Pittsburgh.

Baluja, S., and S. Davies. 1997. Combining multiple optimization with optimal dependency trees. *Technical Report CMU-CS-97-157.* Carnegie Mellon University, Pittsburgh.

Beizer, B. 1990. *Software Testing Techniques.* Van Nostrand Rheinhold.

Burton, S. 2000. Automated Testing from Z Specifications. *Technical Report YCS329.* Department of Computer Science, University of York, Heslington, York.

Campos, V., F. Glover, M. Laguna and R. Martí. 2001. An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. *Journal of Global Optimization* 21:397-414.

Castillo, E., J. Gutiérrez, and A. Hadi. 1997. *Expert Systems and Probabilistic Network Models.* Springer-Verlag.

Chow, C., and C. Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on information Theory* 14:462-467.

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

De Bonet, J. S., C. L. Isbell, and P. Viola. 1997. MIMIC: Finding Optima by Estimating Probability Densities. In *Advances in Neural Information Processing Systems 9*, 424-430. MIT Press.

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

Duran, J. W., and S. C. Ntafos. 1984. An evaluation of random testing. *IEEE Transactions on Software Engineering* 10:438-444.

Fenton, N. E. 1985. The structural complexity of flowgraphs. In *Graph Theory with Applications to Algorithms and Computer Science*, eds Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, C. E. Wall, 273-282. John Wiley & Sons.

Glover, F., and M. Laguna. 1997. *Tabu search.* Kluwer Academic Publishers.

Glover, F., M. Laguna, and R. Martí. 2000. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics* 39(3):653-684.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

Henrion, M. 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proceedings of the Second Conference on Uncertainty in Artificial Intelligence*, 149-163. North-Holland.

Jones, B., H. Sthamer, and D. Eyres. 1996. Automatic structural testing using genetic algorithms. *Software Engineering Journal* 11(5):299-306.

Kirkpatrick, S., C. D. Gellat, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220:671-680.

Korel, B. 1990. Automated software test data generation. *IEEE Transactions on Software Engineering* 16(8):870-879.

Laguna, M., and R. Martí. 2003. *Scatter Search. Methodology and Implementations in C.* Kluwer Academic Publishers.

Larrañaga, P., R. Etxeberria, J. A. Lozano, and J. M. Peña. 2000. Combinatorial optimization by learning and simulation of Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 343-352. Morgan Kaufmann.

Larrañaga, P., and J. A. Lozano. 2002. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers.

Michael, CC., and G. McGraw. 2001. Generating software test data by evolution. *IEEE Transactions on Software Engineering* 27(12):1085-1110.

Mühlenbein, H., and G. Paaß. 1996. From recombination of genes to the estimation of distributions I. Binary Parameters. In *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, 178-187. Springer.

Mühlenbein, H. 1998. The equation for response to selection and its use for prediction. *Evolutionary Computation* 5(3):303-346.

Pargas, R., M. Harrold, and R. Peck. 1999. Test-Data Generation Using Genetic Algorithms. *Journal of Software Testing, Verification and Reliability* 9(4):263-282.

Pelikan, M., D. E. Goldberg, and F. Lobo. 1999. A survey of optimization by bulding and using probabilistic models. *Technical Report IlliGAL Report 99018.* University of Illinois at Urbana-Champaing, Urbana.

Sagarna, R., and J.A. Lozano. 2003. On the performance of Estimation of Distribution Algorithms applied to Software Testing. *Technical Report EHU-KZAA-IK-1/03.* The Univertsity of the Basque Country, Spain.

Sagarna, R., and J.A. Lozano. 2003. Variable Search Space for Software Testing. to appear in *Proceedings of the IEEE International Conference on Neural Networks and Signal Processing.*

Sthamer, H. 1996. The automatic generation of software test data using genetic algorithms. PhD Thesis. University of Glamorgan, Pontyprid, Wales, Great Britain.

Tracey, N., J. Clark, K. Mander, and J. McDermid. 1998. An automated framework for structural test-data generation. In *Proceedings of the 13th IEEE Conference on Automated Software Engineering*, 285-288. IEEE CS Press.

Wegener, J., A. Baresel, and H. Sthamer. 2001. Evolutionary test environment for automatic structural testing. *Information and Software Technology* 43:841-854.