Revista de Sistemas de Informação da FSMA
n. 5 (2010) pp. 3-13

SISTEMAS DE INFORMAÇÃO

http://www.fsma.edu.br/si/sistemas.html

TRILHA ESTUDANTIL

Optimization in Software Testing Using Metaheuristics

Fabrício Gomes de Freitas[1], Camila Loiola Brito Maia[2], Gustavo Augusto Lima de Campos[3].
Jerffeson Teixeira de Souza[4]

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Avenida Paranjana, 1700 – Fortaleza, CE – Brasil
fabriciogf@uece.br, camila.maia@gmail.com, gustavo@larces.uece.br, jeff@larces.uece.br

***Abstract*— There are Software Test problems that may not be solved with traditional software engineering techniques.Nevertheless, such problems may be modeled mathematically in order to be solved with mathematical optimization, specially with the use of metaheuristics. In this context, a new research field called Search based Software Engineering (SBSE), which deals with solving software engineering problems by means of optimization techniques, has emerged. Given the significance of the Software Testing phase, a specific subarea called Search Based Software Testing (SBST) has become increasingly important. This paper presents the current state of the area and summarizes its future potential. Initially, we describe the main metaheuristics techniques used in the area. We follow with the presentation of the state of the art of SBST through the description of the main problems that have already been modeled and the results achieved. From the results, we can realize the promise of this field.**

***Index Terms*— Software Test, Mathematical Optimization, Metaheurstics.**

I. INTRODUCTION

Software Engineering is extremely relevant for the important stage of systems development. Since its conception in the 1970s, important advances in software quality were achieved due to research models, standards and methodologies to support systems development [1]. The application of such methods takes place in several aspects related to the development process, such as Project Planning, Requirements Analysis and Software Testing.

Unfortunately, in some cases, the conventional methods developed by the specialized scientific community are not able to solve certain problems that arise during the process of software development. These problems occur in inherently complex problems, such as those involving the selection of a solution in a prohibitively large set of possibilities. Problems of such kind require automated resolution methods, so the problem can be solved efficiently.

In the Software Testing phase, we find some problems that can be modeled using automated methods. For example, consider the activity of test cases prioritization. This activity consists in determining the best execution order for the test cases of a system. The quality of an order is defined by means of a coverage metric calculated mathematically that determined how soon a test set covers the entire system, rendering the rest of the tests unnecessary. Although this metric can be described using text, the resolution of the problem itself is too costly given the number of possible permutations among the test cases. Hence, the solution to the problem of test cases prioritization can not be easily described by rules written in textual documents or steps and standards.Nevertheless, the mathematical modeling of the problem as an optimization problem is desirable given the existence of mathematical characteristics in the problem.

[1] Fabrício Gomes de Freitas is Bachelor in Computer Science by State University of Ceará (UECE)

2 Camila Loiola Brito Maia is Master in Computer Science by State University of Ceará (UECE).

3 Dr. Gustavo Augusto Lima de Campos is Professor in Master in Computer Science at Sate University of Ceará (UECE).

4 Jerffeson Teixeira de Souza, PhD, is Professor in Master in Computer Science at Sate University of Ceará (UECE).

It was noticed that in fact it is possible to bring together these two areas of Computer Science in order to allow for the resolution of problems that previously could not be solved satisfactorily. Thus, numerous studies have been conducted in recent years demonstrating the applicability of mathematical optimization techniques in a variety of software engineering problems, especially problems that arise the test phase. Hence, the terms "Search-based Software Engineering (SBSE)" and "Search-based Software Testing (SBST)" were defined to represent these areas. The use of the term "search" is due to the fact that the optimization techniques used in the resolution are known as search algorithms.Specifically, the optimization techniques used are metaheuristics, due to the fact they usually find good solutions to problems, regardless of the particular instance or its size.

This article presents a literature review of the use of metaheuristics in Software Testing problems. This version extends a previous version related to Software Engineering in general [2] in order to provide further details to the work described. Thus, we intend to achieve the following objectives:

- To disseminate this new field of research for software engineers so that they are aware of the possibility of solving complex problems during the development phase;
- Instigating researchers that work on Information Systems to use their expertise and the information presented in this survey to add knowledge or ideas to the area, such as modeling of new problems;
- Work as a knowledge base and guide for future researchers interested in the field, being an overview of the state-of-the-art of the area and indicating which further improvements might be achieved.

The article begins with a concise description of the theory of mathematical optimization, including multiobjective optimization. In addition, we present five metaheuristics widely used in solving SBST problems. Then, we indicate the factors that favor the use of optimization techniques for software houses as a complement to current conventional techniques. After that, the next section presents the current state of research in SBST, including five areas, namely: test data generation, test case selection, test cases prioritization, non functional testing, and functional tests. Finally, the conclusion presents some considerations on the general scenario of applications of metaheuristics in Software Testing and indicates future work in the area.

II. METAHEURISTICS OPTIMIZATION

Optimization intends to maximize or minimize a mathematical function defined by coefficients and variables. The variables that define the function may be restricted, ie, the variables must satisfy a set of equations defined according to each problem instance. Heuristics are a subset of the techniques that can be used to solve optimization problems. Such methods are characterized by being specific to the problem under consideration.

The term metaheuristics was introduced by Glover [3] and represents a class of generic search algorithms. Thus, metaheuristics are generic kind of heuristic, ie, that can be used in different kinds of problems. These methods use ideas from different areas as an inspiration to make the process of finding the solution to optimization problems. As examples of a metaheuristic we can underscore Simulated Annealing [4] which is based on a physical process in metallurgy. Other examples are Genetic Algorithms [5], which are based on concepts of evolution population.

The overall execution process of a metaheuristic is the search for a subset solutions in the solution space guided by fitness functions [6]. The fitnes function is a mathematical function that assigns a value to each solution in the search space.

As a comparison, consider a exhaustive search: to determine the best solution to a problem, all existing solutions are visited and in the end the best is returned. Another type of technique that can be used to solve optimization problems are the exact methods. In this approach the search is done from decisions based on mathematical theorems.

The operation of metaheuristics works in a way that to determine the final solution, only some existing solutions are actually visited. The search is conducted under a process that is specific to each metaheuristic, but it is a way that attempts to intelligently find good solutions. However, there is no guarantee the solution returned by a metaheuristic is the best. The use of metaheuristics can be justified due to some factors, among them [7]:

- Complexity of the internal problem that prevents the application of exact techniques;
- Very large quantity of possible solutions that prevent the use of exhaustive algorithms.

In order to better understand the subject of this work, the following section describes optimization theory in more detail. In addition, five metaheuristics used in the SBST works referenced in this article are explained.

The metaheuristics presented are divided into two sections: single objective optimization and multiobjective optimization. The difference between these classes is just in the amount of functions that can be solved by these techniques. In the case of single objective optimization, the metaheuristics perform the search process with a single satisfaction function. In multiobjective optimization techniques, as the name suggests, the search considers more than one function simmultaneously. The algorithms described below are four single objective (Hill-Climbing, Simulated Annealing, GRASP and GA) and a multiobjective metaheuristic (NSGA-II).

A. Single objective Optimization

In single objective optimization, as stated above, the search for solutions is performed according to the values returnef by a single satisfaction function. If we consider a maximization problem and have a solution with a value greater than another,

the solution with the greater value of the satisfaction function is better than the second. The process continues to follow this concept until some stopping criterion, such as maximum execution time, is satisfied.

1) Hill-Climbing

Hill-Climbing technique is defined as a method of local search solutions [8]. Its name represents the activity performed during the search process: from the current solution the next solution is taken from the local neighborhood so that this next solution is better than the previous one.

The term "climbing" is a reference to maximization problems, where the goal is to find solutions with the highest value to the function of satisfaction. Figure 1 below illustrates this concept.

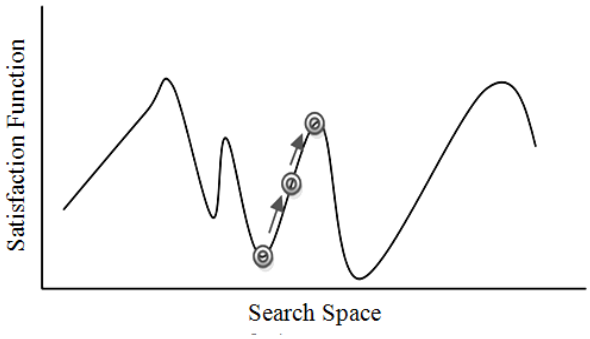

Fig. 1. Process search of Hill-Climbing.

One can see that the Hill-Climbing method provides a simple search strategy. In fact, this is one of the simplest known optimization algorithms. However, this simplicity is a drawback to the search process because this technique is only able to find the local best solution. In other words, the Hill-Climbing method does not explore all the search space, which may have better solutions than the best found in the local space explored.

It is important to stress the fact that the Hill-Climbing method can be used in both maximization and minimization problems. In the minimization case, the algorithm would do the same local search process, but the solutions would be searched for lower values of the function of satisfaction.

2) Simulated Annealing

The metaheuristic Simulated Annealing is considered the best known metaheuristic [4]. Its search procedure is based on an actual physical process that occurs in metals. Specifically, the physical process is taken as the basis originally defined as annealing, or "tempering." Another term used is "cooling" in regard to the activity performed in the process.

In the tempering process, a material is heated to high temperatures and, thereafter, is cooled so that in the end of the process the material is crystallized in a state of minimum energy. In the mathematical optimization inspired technique, the goal is to minimize the value of the function of satisfaction in the solutions visited during the search process.

As the Hill Climbing technique, the Simulated Annealing can be applied both to maximization and minimization problems. If the function should be minimized, the algorithm is ready. If the desire is to maximize the function, we just need to consider its inverse, for the inverse of the minimization of a function corresponds to maximizing it.

The algorithm, unlike the Hill-Climbing technique, allows for the acceptance of solutions that will not improve the value of the satisfaction function. For example, in minimization problems, it is possible to accept the next iteration of a solution with a value that is bigger than the current solution. These acceptances that seem to be contrary to the objective are accepted in order to undertake an intelligent search process. Those acceptances are controlled using functions derived from statistics that are defined for the actual process.

In summary, the algorithm works as follows: if the new solution is better than the current solution, then it is accepted; otherwise, if the new solution is worse, then it is accepted with a certain probability defined in terms of difference between the solutions, the value of temperature and a physical constant. As an illustration, Figure 2 shows the minimization problem process.

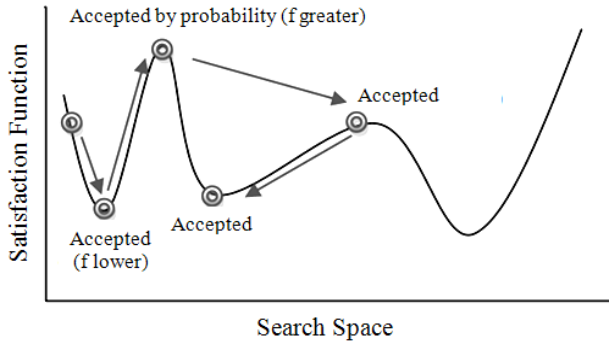

Fig. 2. Search process for Simulated Annealing.

There are several variations of the original algorithm. Among them, one of the most used is the one in which the value of the temperature variable is decreased during the process in order to simulate the cooling process performed in the process. This temperature decrease also decreases the probability of accepting worse solutions during the search process. Thus, the algorithm presents a theoretical convergence to the acceptance of the best solutions in the time limit, while allowing for the diversification of the search by accepting in certain times worse solutions.

3) GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) [9] is a technique that is part of a strategy for a greedy search process. In this metaheuristic, the search is preceded by a build process of adding solution components. The term 'randomized' in the name of this algorithm represents the randomness of the choice of components that may be selected during the construction phase. After the construction of a solution, the search process starts. At the end, the solution is compared with

those already found in previous iterations. The best solution in this set is taken as the current solution. Figure 3 below illustrates the algorithm.

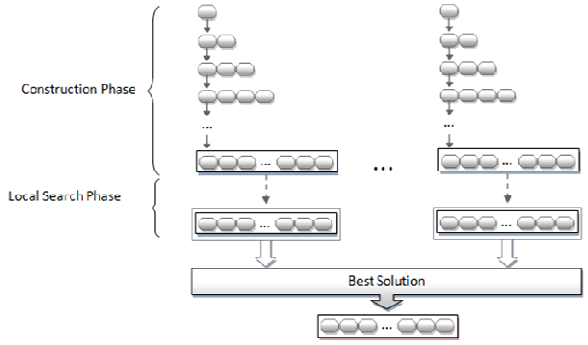

Fig. 3. GRASP operation.

A change in the original GRASP technique is presented in Reactive GRASP [10]. The change in this approach in relation to the original GRASP is in the randomness factor when selecting the components in the construction phase. Thus, during the algorithm execution, the components selection varies between the ability to choose all components or only the best one.

4) Genetic Algorithm

The metaheuristic Genetic Algorithms [5] is a widely used method for solving optimization problems. As its name implies, this technique is based on concepts of genetics, such as population and mutation.

The operation of the metaheuristic uses two genetic operations: crossover, in which the structural information of two solutions are combined in order to generate two new solutions, and mutation, the process by which some random changes may occur in some solutions generated.

After that, the current solutions are evaluated to determine which ones will continue into the next iteration. Thus, the solutions will be selected based on their fitness according to the function of satisfaction. As new solutions are generated from these solutions that have been selected, the process evolves in order to generate increasingly better solutions. In order to allow the process to generate different solutions and to explore different areas of the search space, the operation of mutation is applied with a small rate on the solutions.

B. Multiobjective Optimization

The metaheuristics above are single objective. This means that they solve problems with only one satisfaction function. In these problems the choice of solutions is done simply by comparing the values for each objective considered. For example, if the problem is a maximization problem, a solution with highest function value is better than one with lower value.

Alternatively, the problem can be modeled with more than one function. In these cases, the comparison between solutions is not a trivial process since there is more than one value for comparison. To address this question, the multiobjective optimization approach introduces the the Pareto front concept.

This concept represents a set with the best possible solutions to the problem considering all objectives simultaneously. In other words, this set consists of solutions that are no worse than any other considering all objectives. This concept of being worse or better compared to other solutions, which is known as "dominance", can be used for a formal definition of the Pareto front: the solutions belonging to the Pareto front are those that are not dominated by any other solution, which means they are better solutions than any other in at least one aspect.

1) NSGA-II

The metaheuristic NSGA-II (Non-dominated Sorting Genetic Algorithms II) [11] is a metaheuristic based on multiobjective Genetic Algorithms. The concepts of crossover and mutation are similar to those already presented for the field of GAs.

The NSGA-II starts with the random generation of the set P_0 with N solutions (a population of solutions). Then, a second set Q_0, also of size N, is generated with the use of permutation and mutation operators. These two sets of solutions form the R_0, with population size 2N.

Then R_0 is ordered using the concept of dominance as follows: the solutions that are not dominated by any other are placed in the first Pareto front, F_1; solutions which are dominated by a single solution are placed in the second Pareto front, F_2 and so on until all solutions of R_0 are in a front.

Then, a population P_1 is formed considering the initial Pareto fronts up to the limit of N solutions. When this limit prevents all solutions of a front to be taken, a distance operator is used to select which elements of the current front will be taken to P_1.

This operator aims to select solutions that are not close to others in the search space and create a diverse population of solutions. The population P_1 is then used as a starting point in the next iteration, ie, a population will be formed from the application of genetic operators, and the solutions of the two populations will be ranked according to dominance for the selection of which will participate in the next generation, as shown in Figure 4.

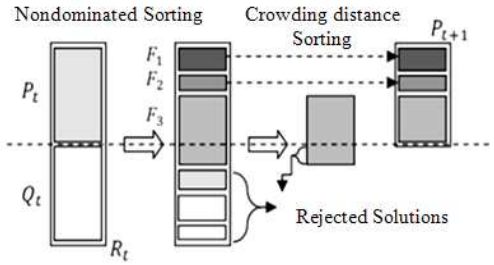

Fig. 4. NSGA-II procedure illustration (Adapted from [11]).

III. Optimization in Software Engineering

Software Engineering, as an engineering discipline, is an area where there are problems that can be solved mathematically [12]. Such problems can be characterized by

the pursuit of a solution in a space of possible solutions. Such problems usually have high number of possibilities and high resolution complexity. Thus, the search process for these problems must be done through an automated method and not through an exhaustive one, allowing those involved in the process to undertake activities where human capacity is more convenient [13] [14].

The first step in that direction came in 1976 [15]. In that work, the goal was to generate test data using numerical maximization. But it was in a second phase, more precisely in 2001, when an important paper in the area [16] coined the term "Search-based Software Engineering (SBSE)."

TABLE I
PHASES AND PROBLEMS TACKLED IN SBSE

Phase	Problems
Requirements	Requirements Analysis Requirements Selection Requirements Planning
Testing	Test Data Generation Selection of Test Cases Prioritization of Test Cases Non-Functional Testing
Estimation	Size Estimation Cost Estimation
Project Planning	Resource Allocation Allocation of Personnel
Code Optimization	Parallelization Compilation Optimization
Maintenance	Re-engineering Software Automated Maintenance
Compiler	Heap allocation Code Size
Software Project	Modularization

From this moment, several researchers in collaboration with software engineers and system developers have modeled and solved various problems of software engineering using search techniques, especially metaheuristics. Table 1 shows the spread of application of metaheuristics to problems in this area.

In this context, one of the Software Engineering areas most prominent in SBSE is Software Testing. This prominence is due mainly to the amount of problems in this area that have already been shown to be prone to modelling and solution using mathematical optimization techniques. A subfield was created specifically for it: Search-based Software Testing (SBST).

At this point, it is important to realize that SBST, as well as SBSE, complements conventional techniques of software engineering already used in systems development. Hence, some problems that were not completely or satisfactorily solved, or not considered by conventional traditional methods, are being studied and solved, which indicates the relevance of the field.

IV. OPTIMIZATION IN SOFTWARE TESTING

This section presents several papers and results in optimization in Software Testing. Among the problems tackled are the test data generation and test cases selection and prioritization. Table 2, indicates some problems already under consideration in SBST, its main characteristics and contributions, along with some main references.

A. Test Data Generation

Test data generation is the process of identifying sets of input data for a program that are valid under the test criteria. The larger and more complex a program, the harder it is to generate such input data. Thus, the use of automated techniques for the generation of these sets seems adequate. In addition, this generation activity has, for example, the goal of achieving the greatest possible amount of testing coverage for the system with the data generated. In this respect, the use of metaheuristics, as optimization techniques, is also appropriate in view of the inability to satisfactorily carry out such activity unless we use mathematical techniques. Otherwise, for example, another possible solution would be to use random generation techniques. However, as already shown [17], such

TABLE II
MAIN SBST PROBLEMS

Problem	Notes	Reference
Test Data Generation	Use of Genetic Algorithms	[17]
	Formalizing the problem	[18]
	Maximizing the paths	[19]
	Number of test times	[20]
	Use of Simulated Annealing	[21]
Test Case Selection	Analysis of five techniques	[22]
	Original multiobjective approach	[26]
	New multiobjective formulation	[27]
	Safe Selection of test cases	[28]
Test Case Prioritization	Prioritization and selection	[26]
	Formalizing the problem	[29]
	Execution time as a factor	[30]
	Comparison of techniques	[31]
	Using the Reactive GRASP	[32]
Non functionl Testing	Test performance	[33]
	Test run time	[34]
	Tests in real context	[35]
Functional Testing	Interation Test	[36]

an approach is not satisfactory.

Korel [18] proposes a test data generator for which the program being tested serves as the input. Thus, the program for which the sets will be generated is executed and its execution is monitored to identify the required data. The program control flow graph is generated and all possible paths are traversed, and finally, the test data input needed to cover such possible paths is generated.

This article has indicated the contribution of the potential application of the optimization problem in the generation of test data. The generic formulation of the problem is shown in Figure 5, where the terms 'n' indicate the points in the code (branchs) and D is the data domain.

$$Let\ P = < n_{k_1}, n_{k_2}, \ldots, n_{k_q} > be\ a\ path$$
$$Find\ x \in D \mid P\ is\ satisfied.$$

Fig. 5. Formalization of the problem of generating test data.

When a path is perceived as different from the control set, and therefore undesided, a minimization function is performed to find corresponding input data values. The author shows that the proposed technique showed significant improvement when compared to other methods.

1) Test Data Generation through evolution

The metaheuristic Genetic Algorithms is also used in data generation [17]. One of the contributions of the referenced work is the use of such a metaheuristic for this problem, since only local search algorithms such as Hill-Climbing have been used before. The approach used in that study, derived from the dynamic generation of test data, is defined by partitioning the program into blocks so that each block can be evaluated as a function according to the input data. The goal is to determine which blocks have the minimum value in the execution so that all points of the code are covered. The results of the study showed that in fact the GA approach is more efficient than random generation, and that as the complexity of the program being tested increased, the improvement achieved by this approach also increases.

2) Improvement of Genetic Algorithms to the problem

Genetic algorithms were used in other studies to generate test data [19]. The objective was to maximize the system flow paths coverage. The authors added features that improved their performance specifically for this problem, such as objective function normalization. Besides proving the effectiveness of the tecnique, the work indicates that with the use of additional techniques it was able to determine the test data with fewer generations (iterations) of the metaheuristic.

3) Simplified approach with Genetic Algorithms

The metaheuristic Genetic Algorithm was also used as a basis for the formalization of an approach to solve this problem [20]. In that work, the concept of chromosomes and evolution are taken from Genetic Algorithms to generate test data. The "individuals" undergo crossover to generate new test data. The population evaluation is made considering the number of appearances of each individual in the test. The authors show that the results obtained with this technique were similar to those of other approaches.

4) Use of Simulated Annealing

Simulated Annealing was also used in the problem of test data generation [21]. The work referenced was carried out in some types of problems of test generation, as in component reuse test. This problem concerns the verification that the new system where the component will be reused has the necessary aspects defined when the component was developed. To validate the approach, systems with conditions that do not allow for the safe reuse of components (ie without errors) were generated. The results showed that the proposed approach was able to generate data to identify system failures that prevent the reuse of components. Moreover, the efficiency of the method for this specific problem was proven, given that the time to find the solution was in the order of seconds.

B. Test Cases Selection

The activity of selecting test cases is the choice of which tests should be performed on a system, whether for the first version or later versions. In the latter case, it is the called regression testing. The latter occurs after the completion of some necessary changes in the system, such as maintenance, which can cause changes in features that were already in operation. In this case, the ideal approach would be to verify all test cases previously used, which would test the whole system again. The need for this solution comes from the fact that the change may have caused some impact on other features of the code. However, there is usually no time or resources to test the whole system again, so it is necessary to select some test cases for execution.

1) Analysis of test cases selection techniques

Mansour et al. [22] compare five algorithms for selecting test cases, namely: Simulated Annealing, Reduction [23], Slicing [24], Dataflow [25], and Firewall [22] regarding the activity of selecting test cases in regression testing.

The satisfaction function used was the code coverage achieved by test cases. The comparison was based on eight quantitative and qualitative criteria: number of test cases, execution time, precision, inclusiveness (how to select cases that cover technical flaws in the new version), processing requirements, type of maintenance, test level and approach type. The results showed that the techniques present better results depending on the criteria. The results indicated that Simulated Annealing found good solutions when measured on the criteria of number of test cases and precision.

2) Multiobjective approach in test cases selection

The concept of Pareto multiobjective optimization is introduced in this problem domain in a 2007 paper [26]. As shown above, the use of multiobjective optimization means that more than one satisfaction function is considered during the process of finding the solutions. Two formulations for the problem were used: the first combines the functions of satisfaction of code coverage and execution cost, and the second version uses the functions of satisfaction of code coverage, cost of implementation and fault history:

Maximize code coverage *Minimize cost*
Maximize code coverage *Minimize cost* *Maximize fault detection*

Fig. 6. Formulations of the test cases selection problem.

Three algorithms were used: a reformulation of the greedy technique, the NSGA-II, and a variation called vNSGA-II. In the experiments, it was used four smaller programs in the suite Siemens (726, 570, 412 and 374 lines of code) and the Space program that has 9564 lines of code.

For the first approach, the metaheuristic NSGA-II achieved the best performance, finding the whole Pareto front, followed by the greedy algorithm and vNSGA-II. However, for the Space program, the greedy technique had the best performance. The vNSGA-II showed similar results to the NSGA, but their results were considerably worse than those achieved by the greedy algorithm.

The results for the second formulation were similar to the ones found with the first one.

3) Test cases selection with customer information

A different approach to the multiobjective problem of test cases selection was presented in 2009 paper [27]. The paper considered more contextual characteristic of the problem, such as the risk of each test case and the importance given by the customer to each requirement that each test case covers. Thus, its main contribution was the participation of the client's vision as an aspect of the activity of selecting test cases.

The proposed multiobjective formulation uses three functions of satisfaction, namely: the minimization of execution time, minimizing risk and maximizing the importance. The article also introduced restrictions to the problem: the time available, and the existence of prerequisites among the test cases. Existence of such prerequisites indicates that some test cases can only be executed after the execution of their predecessors. The complete mathematical formulation is presented in Figure 7.

To solve the problem, the algorithm NSGA-II was used, and a random algorithm was taken as a basis of comparison. For NSGA-II, different population sizes were considered: 10, 20, 30, 40, 50. Populations bigger than 50 found results that were considered similar. The NSGA-II generated better results than the random algorithm, as can be seen in figure 8. The random algorithm generated 50 valid solutions, but only two of them were better than the solutions generated by the NSGA-II. Figure 8 below shows the results of this paper. It is possible to identify the Pareto front found by NSGA-II, with solutions with high importance, low execution time and low risk, and the random solutions dispersed in the search field.

$$1.\ Minimize \sum_{j=1}^{k} executionTime(t_j)$$

$$2.\ Minimize \sum_{j=1}^{k} risk(t_j)$$

$$3.\ Maximize\ import(t_j)$$

$$Subject\ to:$$

$$a)\ \sum_{j=1}^{k} executionTime(t_j) \leq T_{max}$$

$$b)\ \sum_{h=1}^{p} availableTime(p_h) \leq T_{max}$$

$$c)\ \forall r_i \in R, \forall t_j \in T,$$

$$\left(\exists r \in R \ni r \in precedings(r_i)\ and\ cover(t_j, r)\right) \rightarrow t_j \in testCases(r_i)$$

Fig. 7. Formulation to the test cases selection problem with participation of customer insight and constraints.

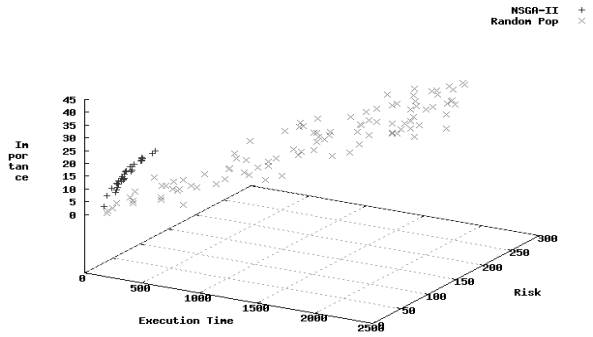

Fig. 8. Results of NSGA-II and random technique for selecting test cases - NSGA-II solutions better than those generated randomly [27].

4) Safe Selection of test cases

Rothermel and Harrold [28] present an approach called safe selection of test cases for regression testing. This concept deals with the selection of test cases in a way that the exclusion of a test case is not allowed if the test case could find faults in the modified version of the system.

The main contribution of the paper is the validation of the technique with several instances of real systems. This allowed the empirical analysis of the results and also for proof of the efficiency of the approach.

C. Test cases prioritization

The activity of test cases prioritization concerns the ordering of test cases so that the most significant ones run earlier, since it is possible that there is not enough time to run all the tests.

In general, the coverage of test requirements is the main objective to be optimized, ie, the ordering of test cases in a way that the maximum number of requirements to be covered at the beginning. This way, if the tests must be interrupted, either for lack of time or resources, one can ensure greater coverage of the test system. The generic formulation of this problem is shown in Figure 9.

$$\begin{aligned}&\text{Let } T \text{ be a Test Suite,}\\&PT \text{ is the set with all permutations}\\&f \text{ is a function } PT \to R.\\&\\&\text{Find } T' \in PT\\&(\forall T''\,(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]\end{aligned}$$

Fig. 9. Formalization of the problem of test cases prioritization.

1) Formalization of satisfaction functions

Early work [29] used greedy algorithms in order to find optimal solutions for the problem. Two of the approaches used were the random one, where a random order is defined, and implementation of test cases without sorting, ie, the order of execution follows the order of creation. The contribution of the initial research was the formalization of the problem and the definition of the satisfaction functions such code coverage, and the probability of fault discovery.

2) Considering the execution time for prioritizing

Walcott et al. [30] propose a technique for prioritizing test cases using Genetic Algorithms using as parameters the code coverage (as generated by the test suite) and the execution time. This approach was compared with the original order approach, where test cases are executed in the order they were written, and also reverse order approach, in which the execution is in the reverse order. The proposed method with Genetic Algorithm outperformed the greedy techniques and purely random approaches.

3) Relation between test cases prioritization and selection

Yoo and Harman [26] described an approach using the concept of Pareto multiobjective optimizatin for test cases prioritizing and selecting.

The objectives used were code coverage, the cost of implementation and the history of fault detection. The specific objective was to find a priority vector of decision variables that represent the ordering of test cases. Other algorithms were compared with the proposed algorithm and the paper concluded that the Genetic Algorithm outperformed the Greedy Algorithm in the case of two satisfaction functions, and in all other situations the greedy algorithm had the best performance.

4) Comparison of techniques for prioritizing

Li et al. [31] compared five algorithms for prioritizing test cases: Greedy Algorithm, Additional Greedy Algorithm, 2-Optimal Algorithm, Hill-Climbing and Genetic Algorithms. The authors separated small 1,000 suites (8 to 155 test cases) and 1,000 larger suites (228 to 4,350 test cases). Six programs in C language were used in the experiments, each containing between 374 to 11,148 lines of code.

The analysis was performed separately for the small and large programs and the optimization goal was the mean percent coverage of code.

In small suites for small programs, the Genetic Algorithm was slightly better than the others techniques, while the greedy algorithm had the worst performance. However, it was shown from the statistical analysis that there was not significant difference between Genetic Algorithms and additional greedy.

For large programs, no significant difference between the additional greedy algorithm and the 2-optimal was found. The difference between the Genetic Algorithm and the two best techniques (additional greedy and 2-optimal) was low, but significant.

For large test suites, the results were similar to experiments with small ones. In the case of small programs, there was no significant difference between the additional greedy and Genetic Algorithm, and for large programs, no significant difference between the additional greedy and the 2-optimal.

5) Use of Reactive GRASP in prioritizing

The use of the Reactive GRASP for the problem of prioritization of test cases was investigated in [32]. The work performed experiments with several different percentages of the test suites (1%, 2%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100%). The Reactive GRASP was compared with techniques previously reported in the literature, such as greedy algorithm and Genetic Algorithm. Analysis of the results was carried out in terms of both performance and execution time. For comparison's sake, the data used in the experiments were the same as previous work [31]. The satisfaction function used was code coverage. Validating the original work, the Additional Greedy algorithm had the best performance. The Reactive GRASP metaheuristic presented the second best performance, but it was found to be slower than the Additional Greedy.

D. Non-functional testing

Non-functional tests check the system features that are not related to features, such as runtime, usability, security, performance, stress, corporate standards (if it was developed according to standards). These non-functional characteristics are specified by the customer.

Performance tests verify that the application meets the performance requirements, eg, time to display a page, query time for a report, the number of simultaneous accesses, among

others.

Stress tests check how the application behaves when the performance parameters are exceeded in order to understand the behavior of the application in situations of high load or "stress" situation; Their goal is to allow for the planning of the necessary actions so that the impact is minimal. For example, consider that the customer has specified that 20 people could access the application simultaneously. The performance test will be performed as follows: 20 simultaneous accesses are made and, if the application "answers" within the expected time, everything is working correctly. The stress test will use more than 20 accesses in order to see what happens when the application is "stressed".

Another factor considered as non-functional feature is the runtime. In this case, the objective is to identify whether the system responds to an input quickly (before desired) or too slowly (delaying for processing). For this aspect, the non-functional tests focus on the identification of input data that may cause the response time of the system to reach the acceptable limit.

The time is usually calculated in units of processing cycle, bacause, although it may differ between systems, it is more accurate than seconds. The difference between the test run time and performance is that in the first case, one tries to find input data that generate the answer in a desired response time (maximum length specified) and the second measures the response time for a given input generated and compares with the maximum time specified.

1) Performance test

Binad et al. [33] propose a tool called Real Time Test Tool (RTTT) developed using Genetic Algorithm to generate test cases for performance testing. These test cases have inputs that will generate deadlines in the tasks to be performed.

As the authors point out, this is not a simple task to do manually. In the proposed approach, the tests are performed based on a number of activities and each one has a execution priority.

Case studies were conducted with the support of the developed tool. For comparison between experiments, both the number of tasks and their priorities were changed. The results were found to be satisfactory since they offered an automated execution way, but were not compared with other approaches.

2) Runtime test

Wegener et al. [34] were the ones that first proposed the consideration of processor cycles instead of seconds as a run time measurement. The article carried out tests on five systems from different domains to determine the efficiency of the metaheuristic GA compared to random search. The results indicated that in fact the GAs found data within the response times limits.

In another work [35] the authors also conducted experiments in order to compare the results of Genetic Algorithm with random search. The tests were conducted with more parameters to determine the validation of the technique in more realistic contexts. The results showed that Genetic Algorithms were able to find data with extreme values.

E. Functional testing

Functional tests check whether the implementation of an application conforms to its specification, ie, that all features specified by the clients are implemented as the documentation.

1) Interaction Test

Interaction tests were considered in a 2003 article [36], in which the authors recombined various techniques to find a larger set of vectors of coverage that can be expanded or decreased as needed. These vectors are coverage sets within the coverage vectors larger and can be combined to ensure a wider coverage for the system. The authors used the greedy TCG (Test Case Generator) and AETG (Automatic Efficient Test Generator) and metaheuristics Hill Climbing and Simulated Annealing. The metaheuristics algorithms significantly outperformed TCG and AETG on the search for the best covering array. The Simulated Annealing was the best in most experiments, and in some cases this technique found the optimal solution.

V. Conclusion and Future Works

The application of metaheuristics to solve Software Engineering problems is part of a relatively new field called Search Based Software Engineering (SBSE).

In this context, the phase of Software Testing showed an emphasis over other phases of systems development. Thus, the Search-based Software Testing (SBST) subfield was created. The results in this area indicate the potential that this emerging field of research presents. In this sense, this way of viewing the problems of Software Engineering, likewise Software Testing, allows the resolution of problems that were unable to resolve satisfactorily before.

The works described in this article describe the research on problems in SBST, mainly in test data generation, test cases selection and prioritization, functional and non-functional testing.

Based on this literature review study, one can see the importance and impact of this recent field of research. Several studies demonstrate the efficiency and relevance of applying metaheuristics to problems of Software Testing in comparison to random approaches, for example. Moreover, the validity of such application is shown in the ability of the methods in solving problems in the form of optimization problems.

The papers and formulations presented in this study serve as motivation for the perception of contributions in the formulations. Another aspect of the survey is the possibility, presented from the modeling of new problems in the area of Software Testing. As future work, we indicate the modeling and solving of problems in various areas of Software Engineering, particularly in the area of Software Testing. Specifically, the resolution of problems related to the activity of regression testing using metaheuristics that hás not been

used yet.

REFERENCES

[1] T. Dyba, "An empirical investigation of the key factors for success in software process improvement", IEEE Transactions on Software Engineering, IEEE, May 2005, pp. 410-424.

[2] F. G. Freitas, C. L. B. Maia, D. P. Countinho, G. A. L. Campos, J. T. Souza, "Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura", II Congresso Tecnológico Infobrasil (Infobrasil 2009), 2009.

[3] F. Glover, "Future paths for integer programming and links to artificial intelligence", Computer Operational Research, 13, pp. 533-549.

[4] S. Kirkpatrick, D. C. Gellat, M. P. Vecchi, "Optimizations by Simulated Annealing" Science v. 220, pp. 671-680, 1983.

[5] J. Holland, *Adaptation in Natural and Artificial Systems*, 1975.

[6] F. Glover, G. Kochenberger, *Handbook of Metaheuristics*, Springer, 1st edition, 2003.

[7] E. Talbi, *Metaheuristics: From Design to Implementation*, Wiley Publishing, 2009.

[8] C. Blum, A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual analysis", ACM Computing Surveys, Vol. 35, N 3, Setembro de 2003, pp. 268-308.

[9] T. A. Feo, M. G. C. Resende. "Greedy randomized adaptive search procedures", Journal of Global Optimization, 6:109–133, 1995.

[10] M. Prais, C. C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12, 164-176, 2000.

[11] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, V. 6, pp. 182-197, 2000.

[12] M. Harman, "Search-Based Software Engineering", Workshop on Computational Science in Software Engineering, 2006.

[13] J. Clarke, *et al.* "Reformulating software engineering as a search problem", IEE Proceedings Software, Vol. 150, No. 3, Junho de 2003, pp. 161-175.

[14] M. Harman, B. F. Jone, "The SEMINAL workshop: reformulating software engineering as a metaheuristic search problem", ACM SIGSOFT Software Engineering Notes, Volume 26, Issue 6, Novembro de 2001, PP. 62-66.

[15] W. Miller, D. L. Spooner, "Automatic generation of floating-point test data", IEEE Transactions on Software Engineering, IEEE, 1976, pp. 223-226.

[16] M. Harman, B. F. Jones, "Search-based software engineering", Information and Software Technology, 2001, pp. 833-839.

[17] C. C. Michael, G. McGraw, M. Schatz, "Generating software test data by evolution", Proceedings of IEEE Transactions on Software Engineering, Vol. 27, Number 12, 2001, pp. 1085-1110.

[18] B. Korel, "Automated software test data generation", Proceedings of IEEE Transactions on Software Engineering, Vol. 16, Number 8, 1990, pp. 870-879.

[19] I. Hermadi, M. A. Ahmed, "Genetic algorithm based test data generator", The 2003 Congress on Evolutionary Computation, 2003.

[20] S. Khor, P. Grogono, "Using a Genetic Algorithm and formal concept analysis to generate branch coverage test data automatically", 19th International Conference on Automated Software Engineering, 2004, pp. 346-349.

[21] N. Tracey, J. Clark, K. Mander, J. McDermid, "An automated framework for structural test-data generation", Proceedings of the International Conference on Automated Software Engineering, pp. 285-288.

[22] N. Mandour, R. Bahsoon, G. Baradhi, "Empirical comparison of regression test selection algorithms", The Journal of Systems and Software 57, 2001, pp. 79-90.

[23] M. J. Harrold, R. Gupta, M. L Soffa, "A methodology for controlling the size of a test suite", ACM Transactions on Software Engineering and Methodology, Vol. 2, Issue 3, 1993, pp. 270-285.

[24] H. Agrawal, J. R. Horgan, E. W. Krauser, S. London, "Icremental regression testing", Conference on Software Maintenance, 1993, pp. 348-357.

[25] R. Gupta, M. J. Harrold, M. L. Soffa, "Program Slicing-Based regression testing techniques", Software Testing, Verification and Reliability, Vol 6, Number 2, 1996, pp. 83-111.

[26] S. Yoo, M. Harman, "Pareto efficient Multi-Objective test case selection", Proceedings of the International Symposium on Software Testing and Analysis, 2007, pp. 140-150.

[27] C. L. B. Maia, R. A. F. Carmo, F. G. Freitas, G. A. L. Campos, J. T. Souza, "A Multi-Objective approach for the regression test case selection problem", XLI Simpósio Brasileiro de Pesquisa Operacional, 2009.

[28] G. Rothermel, M. J. Harrold, "Empirical studies of a safe regression test selection technique", IEEE Transactions on Software Engineering, vol. 24, no. 6, Agosto de 1998, pp. 401-419.

[29] G. Rothermel, R. J. Untch, C. Chu, "Prioritizing test cases for regression rest", IEEE Transactions on Software Engineering, vol. 7, no. 10, 2001, pp. 929-948.

[30] K. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos, "Time-Aware test suite prioritization", Proceedings of the International Symposium on Software Testing and Analysis, 2006, pp. 1-12.

[31] Z. Li, M. Harman, R. M. Hierons, "Search algorithms for regression test case prioritization", IEEE Transactions on Software Engineering, vol. 33, no. 4, Abril de 2007, pp. 225-237.

[32] C. L. B. Maia, R. A. F. Carmo, F. G. Freitas, G. A. L. Campos, J. T. Souza, "Automated test case prioritization with reactive GRASP", Advances in Software Engineering, vol. 2010, 2010, pp. 29-46.

[33] L. C. Brinad, Y. Labiche, M. Shousha, "Performance stress testing of Real-Time systems using Genetic Algorithms", Technical Report, Carleton University, 2004.

[34] J. Wegener, H. Sthamer, B. F. Jones, D. E. Eyres, "Testing real-time systems using Genetic Algorithms", Software Quality Control 6 (2) 127–135, 1997.

[35] J. Wegener, M. Grochtmann, "Verifying timing constraints of real-time systems by means of evolutionary testing", Real-Time Systems 15 (3) (1998) 275–298.

[36] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, "Constructing test suites for interaction testing", Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 38–48.