A Constrained Particle Swarm Optimization Approach for Test Case Selection

Luciano S. de Souza, Ricardo B. C. Prudêncio, Flavia de A. Barros
Center of Informatics (CIn), Federal University of Pernambuco
CEP 50732-970 - Recife (PE) - Brazil
\{iss2, rbcp, fab\}@cin.ufpe.br

Abstract

Automatic Test Case selection is an important task to improve the efficiency of Software Testing. This task is commonly treated as an optimization problem, whose aim is to find a subset of test cases (TC) which maximizes the coverage of the software requirements. In this work, we propose the use of Particle Swarm Optimization (PSO) to treat the problem of TC selection. PSO is a promising optimization approach which was not yet investigated for this problem. In our work, PSO was used not only to maximize coverage of requirements, but also to consider the cost (execution effort) of the selected TCs. For this, we developed a constrained PSO algorithm in which execution effort was treated as a constraint in the search, whereas the requirements coverage is used as the fitness function. We highlight that, although execution effort is an important aspect in the test process, it is generally neglected by the previous work that adopted search techniques for TC selection. In experiments performed in different test suites, PSO obtained good results when compared to other search techniques.

1 Introduction

In the search for quality, Software Testing has become a central task in the software development process, aiming to assure quality and reliability [1]. However, this task can be very expensive, taking up to 50% of the total software development cost [2]. Several tools to automate testing tasks have been proposed, from test generation to its execution.

Existing tools for the generation of test cases can deliver good test suites concerning requirements’ coverage (e.g., [3, 4]). However, these suites are usually too large to be executed within the available time [5]. Yet, an analysis of these suites reveals redundancies in the test cases, i.e., two or more test cases satisfying the same requirement. Hence, it is desirable to reduce the test suite, in order to respect time constraints, however without compromising coverage.

In this scenario, this work investigated mechanisms to select an adequate subset of a given test suite considering time constraints but safeguarding requirements’ coverage. Test Case (TC) selection is known to be a difficult task, and should not be performed at random. In the absence of automatic tools, this task is usually manually performed in an ad-hoc fashion. Besides being a slow process, manual selection does not always observe requirements’ coverage [6].

Different authors have formulated the TC selection problem as an optimization problem, and tried to solve it by deploying search techniques (e.g., Greedy Search and Genetic Algorithms) [5]. Our work investigated the use of Particle Swarm Optimization (PSO) [7] for TC selection. PSO is a population-based search algorithm inspired in the social behavior of bird flocks. An increasing attention has been given to PSO in recent years, motivated by its success in different problems when compared to Genetic Algorithms and other search techniques [8]. Despite that, PSO was not yet investigated in the context of TC Selection.

Regarding selection criteria, the majority of the related previous works only deploy requirements’ coverage; the cost of executing the TCs was rarely considered. We implemented a Binary Constrained PSO (BCPSO) [9, 10] considering both the requirements’ coverage (quality) and the execution effort (cost) of the selected subset of TCs. We used requirements’ coverage as the fitness function to be optimized, and used execution effort as a threshold constraint in the search. We also propose a hybrid PSO algorithm which integrates a greedy search mechanism to refine the solutions provided by BCPSO.

The proposed algorithms were evaluated using two different real-world test suites of functional TCs related to the mobile devices domain. The performed experiments obtained very satisfactory results. This work is being developed with the support of the CIn-Motorola Brazil Test Center (CIn-BTC) Research Group [3], a partnership between CIn/UFPE and the Motorola Industrial Ltda (Brazil).

Section 2 briefly discusses the Test Case Selection problem. Section 3 presents the proposed approach. Section 4 brings the experiments and obtained results. Finally, section 5 presents some conclusions and future work.
2 Test Case Selection

According to [6], Test Case selection is the task of selecting a subset of TCs from a test suite in order to achieve a given goal (e.g., maximum test suite size, minimum desired coverage etc). TC selection is commonly treated as an optimization problem by the literature [5]. In this context, a search technique explores the space of possible subsets of test cases aiming to find the subset which best matches the problem’s goal. However, the selected subset may not have the same coverage as the original test suite.

The TC selection problem is NP-Complete [5], in such a way that it is difficult to solve it in a deterministic way. Hence, heuristic search approaches are more feasible. Among the search algorithms used in this context, we can cite the HGS algorithm [11], the GRE algorithm [12], Greedy Search heuristics for set-covering [13] and Genetic Algorithms [14].

The criterion of coverage requirement to be optimized depends on the type of Software Testing. For structural test, requirements are “pieces” of program code (e.g., blocks, statements, decisions) that should be exercised by the test cases [15, 11]. For functional test, in turn, the requirements are reported in a specification of the software [6].

Although previous works have focused on optimizing the requirements coverage of the selected subset of TCs, the cost of executing these tests is also important. According to [14], the actual aim of TC selection is to achieve better efficiency in terms of cost. Nevertheless, few works were proposed in the context of test case selection considering cost [14, 16]. Yet, these works performed experiments only with structural test (although they aim to cover functional TC as well). In fact, it is harder to consider execution cost of functional tests, since it is necessary to estimate the cost of manually typing the TC before its execution [17].

The following sections presents details of our approach, which considers both coverage and cost when selecting TCs. Our experiments evaluated the proposed algorithms in two functional test suites.

3 PSO to Test Case Selection

Particle Swarm Optimization (PSO) [7] algorithm is a population-based search technique inspired on the birds flocks. The basic PSO algorithm initially defines a random population of particles, each one having a position in the search space and a velocity. The position codifies a candidate solution for the problem being solved and the velocity indicates the direction of the search performed by the particle. The particles are evaluated by a fitness function to be optimized. For a number of iterations, the particles fly through the search space, being influenced by each one’s own experience and by the experience of their neighbors. Particles change position and velocity continuously, aiming to reach a better position.

PSO has shown to be a simple and efficient algorithm compared to other search techniques, including for instance the widespread Genetic Algorithms [8]. In the current work, we investigated the use of PSO to the problem of TC selection. We can point out some applications of PSO in Software Testing, particularly for test case generation (e.g., [18]). However to the best of our knowledge, there is no work investigating PSO for the TC selection problem.

In our proposal, we formulated the TC selection problem as a constrained optimization task in which requirements’ coverage is the fitness function to be optimized, and the execution effort of the selected TCs is used as a constraint in the search. As said before, previous works that deployed search techniques to TC selection rarely considered the execution cost of the selected TCs. Hence, the investigation of a constrained TC selection problem, and the use of constrained PSO to solve it are also specific contributions of our work.

We developed a Binary Constrained PSO (BCPSO) by merging two versions of PSO: (1) the binary version of PSO proposed in [9], since the TC selection problem under consideration has a binary search space; and (2) the PSO version which deals with constrained problems, proposed in [10]. The BCPSO is presented in section 3.1.

Furthermore, a hybrid implementation of BCPSO with a greedy-search algorithm, named as Forward Selection (FS), was developed aiming to verify whether some improvement in PSO performance could be obtained by using a local search mechanism for each particle (see section 3.2).

As it will be seen in section 4, the proposed algorithms were applied to TC Selection of functional test. We highlight that no previous work was found in the literature that performed cost-constrained TC Selection in the context of functional software test.

3.1 The Binary Constrained PSO

In this section, we provide a description of the BCPSO algorithm, developed to select test cases considering both coverage of software requirements and execution effort.

3.1.1 Problem Formulation and Fitness Function

A particle position indicates a candidate subset of TCs to be used in the software testing process. Given a test suite $T = \{T_1, \ldots, T_n\}$ of $n$ test cases, a particle position is represented as a binary vector $t = (t_1, \ldots, t_n)$, in which $t_j \in \{0, 1\}$ indicates the presence (1) or absence (0) of the test case $T_j$ among the subset of selected TCs.

The fitness of each particle is the percentage of requirements covered by the represented solution. Formally, let...
Let $R = \{R_1, \ldots, R_k\}$ be a given set of $k$ requirements. Let $F(T_j)$ be a function that returns the subset of requirements in $R$ covered by the individual test case $T_j$. Then, the fitness function of a particle represented by $t_j$ is given as:

$$\text{Fitness}(t_j) = 100 \times \frac{|\bigcup_{j=1}^k F(T_j)|}{k} \quad (1)$$

In eq. (1), $\bigcup_{j=1}^k F(T_j)$ is the union of requirements subsets covered by the selected test cases (i.e., $T_j$ for which $t_j = 1$).

As said, the execution effort of the selected TCs is used as a constraint in the search process. Formally, each test case $T_j \in T$ has a cost score $c_j$. The total cost of a solution $t$ is then defined as:

$$\text{Cost}(t) = \sum_{j=1}^{t_j} c_j \quad (2)$$

In our work, the cost $c_j$ was computed for each test case by using the execution effort estimation model developed by Aranha and Borba [17]. Finally, we formulated the optimization problem solved by PSO as:

$$\text{maximize} : \quad \text{Fitness}(t) \quad (3)$$

$$\text{subject to} : \quad \text{Cost}(t) < \theta \quad (4)$$

In the above equation, $\theta$ is a threshold given by the user, which reflects the problem constraint: the amount of resources available to perform the Software Testing. In the constrained PSO proposed in [9], if a particle violates the constraint (i.e., if it is an infeasible solution), its fitness is penalized. The penalization of the fitness function was defined in our work as follows:

$$\text{Fitness}_{\text{penalty}}(t) = \text{Fitness}(t) - 100 \quad (5)$$

For particles violating the problem constraint, the fitness values initially computed using eq. (1) are replaced by the values computed by the penalized function in eq. (5), assuming non-positive values. This way, the infeasible particles do not influence the other particles of the population.

### 3.1.2 Search Operation

In PSO, each particle explores the search space by updating its position according to a velocity vector $v = (v_1, \ldots, v_n)$ which indicates the direction of the search performed by the particle. The velocity vector is updated at each PSO iteration by the following equation:

$$v \leftarrow \omega v + C_1 r_1 (t - t) + C_2 r_2 (g - t) \quad (6)$$

In eq. (6), $t$ indicates the best position achieved by the particle, and $g$ is the best position achieved by its neighbors. $r_1$ and $r_2$ are random values in the interval $[0,1]$. The first part of eq. (6) represents the inertia factor, the second one represents the cognitive component of the search (own experience) and the third one represents the social component of the search (experience from the neighborhood).

The parameters $\omega$, $C_1$ and $C_2$ control the trade-off between the cognitive and the social behavior of the particles. In our work, $\omega$ linearly decreases from 0.9 to 0.4, and $C_1 = C_2 = 2$ (as suggested by [19]). Finally, in order to define neighborhood, each particle is indexed, and its neighborhood consists of the particles which are its predecessor and successor.

In PSO, the particle position is updated in the direction of its velocity. In BCPSO, the update of positions followed the same operations originally proposed in the binary PSO [9]. First, the sigmoid function is used to normalize the values of velocity in the interval $[0,1]$ as follows:

$$\text{sig}(v) = \frac{1}{1 + e^{-v}} \quad (7)$$

Finally, the new particle position is updated as follows:

$$t_j = \begin{cases} 
1, & \text{if } r_j \leq \text{sig}(v_j) \\
0, & \text{otherwise}
\end{cases} \quad (8)$$

In the above equation, $r_j$ is a random number sampled in the interval $[0,1]$. Equation (8) was proposed in [9] in order to certify that the new positions are still binary vectors. The position value $t_j$ tends to 1 when the velocity assumes higher values (closer to 1). In its turn, $t_j$ tends to 0 for lower values of velocity ($v_j$ close to 0).

### 3.2 The Hybrid BCPSO-FS

A greedy search algorithm, named as Forward Selection (FS) inspired on [20], was adapted to consider the execution effort constraint. The search process starts with an empty set of TCs, and progressively selects a TC that, when added to the TCs set, does not violate the cost constraint. At each step, the algorithm chooses the non-selected test case $T_j \in T$ with highest requirements coverage ($F(T_j)$), and verifies whether the cost constraint is not violated. Whenever a tie is observed between the best remaining TCs, a random choice is made. This process is repeated until all remaining TCs will cause a violation of the cost constraint when individually added to the TCs subset being formed.

The FS algorithm is combined to the BCPSO in order to create a new hybrid algorithm (named here as BCPSO-FS). The hybrid algorithm is similar to BSPSO, however, at each iteration of BCPSO, the FS algorithm is used as a local search mechanism in order to refine each particle’s solution. Each particle produced in a BCPSO iteration is given as an initial solution to FS, which refines it until the stopping criterion of FS is reached. The solutions optimized by FS are
4 Experiments and Results

In this section, we present the experiments which evaluated the performance of the implemented search techniques. The experiments were performed to select TCs from two test suites used to test mobile devices\(^1\), referred here as Suite 1 and Suite 2. Both test suites have 80 TCs, however their characteristics are different, as shown in the table 1. Suite 1 is more complex, covering a higher number of requirements. The total effort needed to execute all test cases in Suite 1 is also higher when compared to Suite 2.

The TCs in Suite 1 are less redundant\(^2\), i.e. two distinct test cases rarely cover the same requirement. Hence, for Suite 1, it is expected to be more difficult to find a solution with good fitness evaluation. In Suite 2, in turn, each test case individually covers a higher number of requirements, with a higher level of redundancy. This characteristic makes it easier to select test cases in Suite 2.

In our experiments, we evaluated the search progress of BCPSO and BCPSO-FS for each suite in two different scenarios regarding the threshold (\(\theta\)). In these scenarios, the \(\theta\) threshold was defined to reduce the execution effort to 75% and 50% of the total effort estimated for each suite. The search stopping criterion was set to the maximum number of 200,000 fitness evaluations (which showed to be big enough to allow the convergence of each tested algorithm).

As a basis of comparison, we also performed experiments using a purely random search, which, despite its simplicity, has the advantage of performing a uniform exploration of the search space, being very competitive in other contexts [21]. Each algorithm was executed 100 times for each suite and scenario considered.

Figures 1, 2, 3 and 4 present the average fitness (over the 100 runs) along the number of fitness evaluations. The proposed algorithms outperformed the Random search in all suites and scenarios, which indicates that the proposed algorithms were feasible to optimize the fitness function. By comparing BCPSO to BCPSO-FS, we observed that BCPSO obtained in general a better behavior in the beginning of the search. In the initial iterations, a global explo-

---

\(^1\)These suites were obtained from the Motorola CIn-BTC (Brazil Test Center) research project

\(^2\)Redundancy of a suite is measured by the average Jaccard similarity between the sets of requirements covered by each pair of TCs.
ration of the search space performed by BCPSO was more important than the refined search performed by BCPSO-FS. The BCPSO-FS, in turn, obtained better results when used to refine relatively good regions of the search space. In these regions, the FS local search could achieve positions which were difficult to be yielded by BCPSO.

As expected, better results were observed for Suite 2 (the less redundant suite) for which the search algorithm converged to adequate solutions within a lower number of fitness evaluations. We could also notice that the constraint imposed in the search had an impact in the results. For $\theta=75\%$, the search algorithms obtained in general better results, since this constraint is not so strong. In fact, the higher is the value of $\theta$, the higher is the number of subsets of TCs that do not violate the execution effort constraint (i.e. the higher is the number of feasible solutions). These results indicate that the performance obtained by the search algorithms would be dependent both on the characteristics of the test suite given as input, and on the resources constraints imposed by the context of application.

Finally, we compared the best solutions found by each algorithm BCPSO, BCPSO-FS, FS and Random (see table 2). Our results showed that the hybrid BCPSO-FS outperformed all other algorithms in all scenarios for Suite 1 (the most complex one). For Suite 2, all algorithms (apart from Random) achieved coverage values of 100%. Despite its simplicity, the FS algorithm has shown to be very competitive, particularly when compared to the BCPSO. The hybridization showed to improve the BCPSO algorithm by adding an explicit mechanism of local search. Other hybrid search strategies can be investigated in future work, for instance, by adopting other greedy search techniques.

5 Conclusion

In this work, we investigated the use of Particle Swarm Optimization for TC selection. We can point out some contributions of the current work. First, to the best of our knowledge, PSO was not investigated yet in the context of TC selection. Also, we considered the effort in executing the selected test cases by formulating TC selection as a constrained optimization task and by proposing specific versions of PSO to treat this task.

We implemented a Binary Constrained PSO (BCPSO) and a hybrid algorithm BCPSO-FS, which integrated the FS algorithm to improve the performance of the BCPSO. In the performed experiments, BCPSO and BCPSO-FS outperformed the results obtained by a Random search, thus verifying the viability of using PSO to TC selection. The BCPSO-FS obtained good results when compared to its individual components. The good performance of BCPSO-FS was achieved by combining the global search performed by
BCPSO and the local search performed by the FS algorithm.

Several extensions of the current work can be considered in the future. First, we intend to perform experiments on more test suites, and investigate the effect of the PSO’s parameters in its performance. Following, we intend to adapt a higher number of algorithms for dealing with constrained TC selection and to perform a more complete comparison of techniques. Finally, we will investigate new strategies to combine search techniques, in order to provide more robust hybrid algorithms for TC selection.

Acknowledgments: This work was supported by the National Institute of Science and Technology for Software Engineering (INES³), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

References


³www.ines.org.br