Expert Systems with Applications 40 (2013) 4887–4896

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Search based constrained test case selection using execution effort

Luciano S. de Souza [a], Ricardo B.C. Prudêncio [a,*], Flavia de A. Barros [a], Eduardo H. da S. Aranha [b]

[a] Center of Informatics (CIn), Federal University of Pernambuco, Recife – PE, Brazil
[b] Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal – RN, Brazil

ARTICLE INFO

Keywords:
Search Based Test Case Selection
Software Testing
Execution effort
Search based software engineering

ABSTRACT

Software testing is essential to guarantee high quality products. However, it is a very expensive activity, particularly when manually performed. One way to cut down costs is by reducing the input test suites, which are usually large in order to fully satisfy the test goals. Yet, since large test suites usually contain redundancies (i.e., two or more test cases (TC) covering the same requirement/piece of code), it is possible to reduce them in order to respect time/people constraints without severely compromising coverage. In this light, we formulated the TC selection problem as a constrained search based optimization task, using requirements coverage as the fitness function to be maximized (quality of the resultant suite), and the execution effort (time) of the selected TCs as a constraint in the search process. Our work is based on the Particle Swarm Optimization (PSO) algorithm, which is simple and efficient when compared to other widespread search techniques. Despite that, besides our previous works, we did not find any other proposals using PSO for TC selection, neither we found solutions treating this task as a constrained optimization problem. We implemented a Binary Constrained PSO (BCPSO) for functional TC selection, and two hybrid algorithms integrating BCPSO with local search mechanisms, in order to refine the solutions provided by BCPSO. These algorithms were evaluated using two different real-world test suites of functional TCs related to the mobile devices domain. In the performed experiments, the BCPSO obtained promising results for the optimization tasks considered. Also, the hybrid algorithms obtained statistically better results than the individual search techniques.

1. Introduction

The demand for high quality products has imposed a growing emphasis on testing activities in the software development process (Beizer, 1990). However, Software Testing is a very expensive activity, sometimes reaching 40% of the final development cost (Ramler & Wolfmaier, 2006). In this scenario, automation seems to be the key solution for improving the efficiency and effectiveness of the testing process.

We can identify in the related literature several techniques and tools aimed at the (partial or total) automation of the testing tasks, ranging from test generation to its automatic execution. Usually, each technique/tool focuses on only one of the main software testing approaches: either White Box (structural) or Black Box (functional) testing. Regardless the approach, the testing process relies on the (manual or automatic) generation and execution of a Test Suite (TS). A TS comprises a set of Test Cases (TC), each one consisting of a set of inputs/actions to test the software, execution preconditions, and a pass/fail condition (expected results).

When analyzing the existing tools for automatic TC generation, we observe that they usually deliver very large test suits, trying to cover all possible scenarios (e.g. Borba, Torres, Marques, & Wetzel, 2007; Kissoum & Sahnoun, 2007). The aim is to provide a good coverage of the adopted test adequacy criterion (e.g., code coverage, requirements coverage) in order to satisfy the test goal. Note that manually created test suits may also be large, for the same reason mentioned above.

Although it is desirable to fully satisfy the test goal, the execution of large suites is a very expensive task, demanding a great deal of the available resources (time and people) (Harold, Gupta, & Soffa, 1993). However, large test suites usually contain redundancies regarding the adopted test adequacy criterion (i.e., two or more TCs covering the same requirement/piece of code). Therefore, it is possible to reduce the test suite in order to respect time/people constraints, however without severely compromising coverage. The task of reducing a test suite based on a given selection criterion is known as *Test Case selection* (Borba, Cavalcanti, Sampaio, & Woodcock, 2007). Notice that the test selection criterion is dependent upon the adopted test adequacy criterion.

* Corresponding author.
E-mail addresses: lss2@cin.ufpe.br (L.S. de Souza), rbcp@cin.ufpe.br (R.B.C. Prudêncio), fab@cin.ufpe.br (F.d.A. Barros), eduardoaranha@dimap.ufrn.br (E.H.d.S. Aranha).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.02.018

In this light, our work investigated strategies to select an adequate subset of a given test suite having as selection criterion the functional requirements coverage (quality of the resultant suite), and the execution effort (time/cost of the selected subset) as a constraint.

TC selection is known to be a hard task, since there may be a huge number of TC combinations to consider when searching for an adequate TC subset. In this scenario, manual TC selection does not seem to be a good choice. Besides being a slow process, manual selection is usually an *ad hoc* process, relying too much on the test engineer's previous knowledge. This way, it does not guarantee to preserve coverage of the adopted test adequacy criterion (Cartaxo, Machado, & Oliveira Neto, 2009). Thus, manual TC selection is generally adopted when there are no available tools to automatically perform this task.

When analyzing the works on automatic TC selection, we identify different techniques for systematically performing this task. First, we cite heuristic and greedy solutions (see Borba et al., 2007; Chen & Lau, 1998; Cartaxo et al., 2009; Harold et al., 1993; Lin & Huang, 2009), which provide potentially good TC subsets regarding the coverage of the test adequacy criterion. However, these non-optimization strategies are not always applicable when dealing with large test suites, since they are computationally expensive (Yoo & Harman, 2010; Lin & Huang, 2009).

A very promising approach to treat the TC selection problem relies on the use of optimization search techniques (Barltrop, Clement, Horvath, & Lee, 2010; Mansour & El-Fakih, 1999; Ma, Sheng, & Ye, 2005; Souza, Prudencio, & Barros, 2010; Yoo & Harman, 2007). The aim here is to select a TC subset that optimizes a given objective function (i.e., the given selection criterion). We highlight here the work of Yoo and Harman (2010), Yoo and Harman (2007), which uses Genetic Algorithms for structural TC selection.

Our work formulated the TC selection problem as constrained optimization task, where requirements coverage is the fitness function to be optimized, and the execution cost (time) of the selected TCs is used as a explicit constraint in the search process. We investigated the use of Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995) algorithm for this problem. PSO is a population-based search algorithm inspired by the social behavior of bird flocks.

PSO has shown to be a simple and efficient algorithm when compared to other search techniques, including for instance the widespread Genetic Algorithms (Hodgson, 2002; Eberhart & Shi, 1998). Despite that, besides our previous works (Souza et al., 2010, Souza, Miranda, Prudencio, & Barros, 2011), we did not find in the related literature any other work using PSO for TC selection. Therefore, it is worthwhile to further evaluate the use of PSO for this task.

We implemented a Binary Constrained PSO (BCPSO) (Hu & Eberhart, 2002; Kennedy & Eberhart, 1997) and two hybrid PSO algorithms, which integrate BCPSO with local search mechanisms to refine the solutions provided by the BCPSO. The proposed algorithms were evaluated using two different real-world test suites of functional TCs related to the mobile devices domain.[1] The BCPSO-FS hybrid algorithm obtained statistically better or at least equivalent results than the individual search techniques.

In our preliminary work (Souza et al., 2010), we performed some experiments to evaluate the BCPSO for constrained TC selection, having obtained promising results. In the current paper, we present the new developments of our research – in particular, the hybrid algorithms-, a more detailed discussion of the conducted case studies, and the statistical analysis of the performed experiments.

As mentioned before, so far we did not find in the related literature any solution considering requirements coverage as test selection criterion together with execution effort (cost) as an explicit constraint for functional test case selection. The previous works of Elbaum, Malishevsky, and Rothermel (2001) and Walcott, Soffa, Kapfhammer, and Roos (2006)also combine cost constraints with requirements coverage, but they are used for structural test suite prioritization, and the work of Yoo and Harman (2007) for structural test case selection.

In what follows, Section 2 briefly discusses strategies for Test Case selection, and Section 3 brings a detailed presentation of our proposed approach for this problem. Section 4 brings the experiments and obtained results. Finally, Section 5 presents some conclusions and future work.

2. Strategies for Search Based Test Case Selection

As discussed before, although testing is central in the software development process, it is sometimes neglected due to its high cost. A primary way to reduce software testing cost is by reducing test suites.

Given an input test suit, Test Case selection aims to find a relevant subset of TCs regarding some test adequacy criterion (such as the amount of code or functional requirements covered by the chosen TC subset, for instance). Clearly, this task should not be performed at random, since a random choice not always returns a representative TC subset.

A very promising approach for TC selection is to treat this task as a search optimization problem (Yoo & Harman, 2010). In this approach, search techniques explore the space of possible solutions (subsets of TCs), seeking the solution that best matches the test objectives.

When analyzing which search based approach to use, we initially disregard the exhaustive (brute-force) search techniques, since we are facing an NP-complete problem (Lin & Huang, 2009). We also disregard random search since, when dealing with large search spaces, random choices seldom deliver a representative TC subset regarding the adopted test adequacy criterion. In this scenario, more sophisticated approaches, as search based optimization techniques, should be considered to treat this problem.

As known, optimization techniques in general demand the use of one or more fitness functions, which will determine the quality of each possible solution (a TC subset) regarding some chosen search criterion. Each fitness function corresponds to a different search objective.

Regarding single-objective search techniques applied to test case selection, we highlight the use of Simulated Annealing (Mansour & El-Fakih, 1999), conventional Genetic Algorithms (Ma et al., 2005; Mansour & El-Fakih, 1999) and PSO (Souza et al., 2010). Regarding multi-objective techniques, we highlight (Souza et al., 2011; Yoo & Harman, 2010, 2007).

Regardless the technique to use in the selection process, before defining the search objectives, it is necessary to formulate the selection problem considering the chosen test adequacy criterion, which depends on the type of test being performed – whether functional or structural. Structural testing relies on the code structure to derive test cases, which are used to monitor the software behavior during implementation time. On the other hand, functional testing relies on (natural language or formal) specifications of the software, aiming to investigate the behavior of the implemented SW functionalities.

For structural testing, the most usual adequacy criterion is code coverage, which considers the amount of pieces of program code

[1] In order to allow the selection based on requirements coverage, each TC registers/indicates the covered requirements.

(e.g., blocks, statements, decisions) exercised by selected TCs (Harold et al., 1993). On the other hand, functional TC selection usually adopts as selection criterion the amount of functional requirements covered by the TC suites (Borba et al., 2007; Souza et al., 2011).

Besides the test adequacy criterion, another issue to be considered in TC selection is the existence of constraints that must be observed. For instance, in some software testing environments, the test engineers have a restricted time to (manually) execute the test suits. In such cases, it is necessary to incorporate these restrictions as a constraint in the formulation of the search problem. This way, the ordinary search problem becomes a constrained search problem, and the techniques to be used must be able to account for this.

Among existing testing constraints, we highlight the cost (effort) to execute a TC suit, so that it fits within the available time for testing the product. Despite its importance, this constraint has been neglected by researches in the field, mainly due to the difficulties to previously estimate the cost of manually performing each TC in the input suite (Aranha & Borba, 2007). In fact, few works have taken this into account, i.e., have used this measure for TC selection (Malishevsky, Ruthruff, Rothermel, & Elbaum, 2006; Souza et al., 2010, 2011; Yoo & Harman, 2007, 2010).

As said, the work presented here investigates the use of TC execution effort as an explicit constraint in the search process, in order to deliver functional TC subsets respecting the available execution time. Here, we use the *Test Execution Effort Estimation Tool* (Aranha & Borba, 2008) to estimate the time (cost) for manually executing each test case in the input suites before starting the search process (see Section 4.1.1).

Following, Section 3 presents details of our approach for test case selection, and Section 4 brings the experiments performed to evaluate the proposed algorithms.

3. Constrained TC selection based on optimization search techniques

As said, TC selection has been treated by several authors as an optimization task in which a chosen testing criterion has to be optimized. Besides, there are constraints on the testing process to be considered (e.g., the effort to execute the test cases).

In our work, we formulated the TC selection problem as a constrained optimization task in which functional 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 process.

Few works that deploy search techniques to TC selection considered the execution cost of the selected TCs. This is mainly due the difficulty of estimating execution effort for functional test cases. Also, these works do not treat cost constraints in an explicit and systematic way. Yet, besides our previous work (Souza et al., 2010), we did not find any other work on cost-constrained TC selection in the context of functional software testing, as said in Section 1. Hence, we believe that our research work is an original contribution to the area.

We implemented a number of different algorithms for constrained TC selection, including global, local and hybrid search techniques. More specifically, we highlight the use of Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), a global optimization technique which has been poorly investigated in the context of TC selection.

This section is dedicated to present our work in detail. Initially, we have a brief overview of the developed work, including some background information on the PSO algorithm (Section 3.1). Following, Section 3.2 shows the problem formulation in the light of the search optimization approach. Then, Sections 3.3, 3.4, 3.5, 3.6 present the different search techniques investigated here, in order to compare results.

3.1. Overview of the developed work

As said, in this work we investigate the use of PSO applied to the problem of TC selection. PSO has shown to be a simple and efficient algorithm when compared to other search techniques, including for instance the widespread Genetic Algorithms (Eberhart & Shi, 1998; Hodgson, 2002).

We can point out some applications of PSO in Software Testing, particularly for test case generation (Windisch, Wappler, & Wegener, 2007) and regression testing (Kaur & Bhatt, 2011). However, to the best of our knowledge, our previous works (Souza et al., 2010, 2011) are the only ones using PSO for TC selection.

The PSO algorithm is a population-based search technique inspired by the bird 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 parameterized 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. The algorithm stops when a stopping criterion is reached (usually, a predefined number of iterations or number of fitness evaluations).

We implemented seven different algorithms for TC selection, in order to compare results. Initially, we developed a Binary Constrained PSO (BCPSO) by merging two versions of PSO (see Section 3.3): (1) the binary version of PSO proposed in Kennedy and Eberhart (1997), since the TC selection problem under consideration has a binary search space; and (2) the PSO version which deals with constrained problems, proposed in Hu and Eberhart (2002). Following, three local search algorithms were developed: Forward Selection, Backward Elimination and the Hill-Climbing algorithms.

Finally, hybrid implementations of BCPSO were developed by combining it with local search algorithms (namely, the Forward Selection and the Hill-Climbing algorithms). The aim was to verify whether some improvement in PSO performance could be obtained by using a local search mechanism for each particle (see Section 3.5). In different optimization contexts, the combination of global and local techniques has shown performance gains when compared to its individual components (Lvbjerg, Rasmussen, & Krink, 2001 and Juang, 2004).

The developed algorithms were evaluated in an experiment with two case studies using real functional test suites (Section 4). In the experiments, the Tukey HSD multiple comparison analysis (Tukey, 1949) was applied to verify statistical differences among the implemented search techniques.

As said, the work presented here extends our previous work (Souza et al., 2010) with new implementations, putting emphasis on the hybrid algorithms, the statistical analysis of the performed experiments and a more detailed discussion of our case studies. The remaining of this section presents how the Test Case selection problem was formulated, and details of the implemented solutions.

3.2. Problem formulation

In this section, we show how the TC selection task was formulated as a search (optimization) problem in our work. The following formulation was used in all the implemented techniques.

Given a test suite $T = \{T_1, \ldots, T_n\}$ of n test cases, a candidate (subset) solution is represented as a binary vector $\mathbf{t} = (t_1, \ldots, t_n)$, 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 (quality) of a solution is measured as the percentage of requirements covered by it. Formally, 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 solution represented by $\mathbf{t}$ is given by:

$$Fitness(\mathbf{t}) = 100 * \frac{\left|\bigcup_{t_j=1}\{F(T_j)\}\right|}{k} \qquad (1)$$

In Eq. (1), $\bigcup_{t_j=1}\{F(T_j)\}$ 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 $\mathbf{t}$ is then defined as:

$$Cost(\mathbf{t}) = \sum_{j,t_j=1} c_j \qquad (2)$$

In our work, the cost c_j was computed for each test case by using the Test Execution Effort Estimation Tool developed by Aranha and Borba (2008) (see Section 4.1.1 for details).

Finally, we formulated the search (optimization) problem as follows:

$$maximize : Fitness(\mathbf{t}) \qquad (3)$$

$$subject\ to : Cost(\mathbf{t}) < \theta \qquad (4)$$

In Eq. (3), θ is a threshold execution time given by the user, which reflects the search constraint (i.e., the maximum amount of time available to perform the Software Testing).

3.3. The binary constrained PSO

The implemented Binary Constrained PSO (BCPSO) algorithm was developed by merging the binary PSO of Kennedy and Eberhart (1997) and the constrained PSO of Hu and Eberhart (2002). As seen above, each particle in the PSO algorithm has a *position* in the search space and a *velocity*. Each particle explores the search space by updating its position according to a *velocity vector* $\mathbf{v} = (v_1, \ldots, v_n)$, which indicates the direction of the search performed by the particle. The velocity vector is updated at each PSO iteration using the following equation:

$$\mathbf{v} = \omega\mathbf{v} + C_1 r_1(\hat{\mathbf{t}} - \mathbf{t}) + C_2 r_2(\hat{\mathbf{g}} - \mathbf{t}) \qquad (5)$$

In Eq. (5), $\hat{\mathbf{t}}$ indicates the best position achieved by the particle, and $\hat{\mathbf{g}}$ is the best position achieved by its neighbors. r_1 and r_2 are random values in the interval [0,1]. C_1 and C_2 are the acceleration constants. The first term of the right side expression represents the inertia factor, the second term represents the *cognitive* component of the search (own experience), and the third term represents the *social* component of the search (neighborhood experience).

Hence, each particle progressively changes its direction towards the best *global* positions achieved by the neighborhood and the best *local* positions obtained by the particle itself.

The parameters ω, C_1 and C_2 control the trade-off between the cognitive and the social behavior of the particles. In our work, ω linearly decreases from 0.9 to 0.4, and $C_1 = C_2 = 2$ (as suggested by Shi & Eberhart (1998)).

Finally, in order to define neighborhood, the particles in the PSO algorithm are organized in a particular chosen topology that indicates their social structure. In our implementation, we adopted the ring topology (a widespread PSO topology), in a way that the neighborhood of a particle consists solely of its predecessor and its successor in the topology.

In PSO, the particle position is updated according to its velocity. In BCPSO, the update of the particle positions used the same operations originally proposed in the binary PSO (Kennedy & Eberhart, 1997). First, the sigmoid function is used to normalize the velocity values within the interval [0,1] as follows:

$$sig(v_j) = \frac{1}{1 + e^{-v_j}} \qquad (6)$$

Finally, the new particle position is updated as follows:

$$t_j = \begin{cases} 1, & \text{if} \quad r_j \leqslant sig(v_j) \\ 0, & \text{otherwise} \end{cases} \qquad (7)$$

In Eq. (7), r_j is a random number sampled from the interval [0,1]. This equation was proposed by Kennedy and Eberhart (1997) 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).

In the constrained PSO proposed by Kennedy and Eberhart (1997), when a particle violates the constraint (i.e., when it represents an infeasible solution), its fitness is penalized. The fitness function penalization was defined in our work by Eq. (8):

$$Fitness_{Penalty}(\mathbf{t}) = Fitness(\mathbf{t}) - 100 \qquad (8)$$

Whenever a solution violates the problem constraint, the fitness values initially computed using Eq. (1) are replaced by the values computed by the penalized function in Eq. (8), assuming non-positive values.

As it can be observed in Section 4 (Experiments and Results), the BCPSO delivered good results when applied to the TC selection problem. The following section presents the local search techniques which were implemented as a basis of comparison.

3.4. Local search

In our work, we also investigated some well known local search algorithms, aiming to create hybrid algorithms by combining local search with our global search BCPSO algorithm (see Section 3.3). This section presents three local search algorithms adapted to the TC selection problem.

Briefly speaking, local search algorithms choose, at each step, the locally best node (which yields the best fitness) aiming to find the best solution to the problem. The local search algorithms used in our work are the Hill Climbing, the Forward Selection and the Backward Elimination algorithms.[2]

3.4.1. Forward selection

According to Webb (2002), the Forward Selection (FS) technique, also known as Sequential Forward Selection, is a bottom-up search procedure which builds a solution by iteratively adding new nodes to an initially empty set, until a stopping criteria is reached. The current solution is then returned as the solution for the search process.

In our work, the solution is represented by a binary vector $\mathbf{t}$. The algorithm starts with an empty solution (i.e., with all $t_j = 0$), and then performs several iterations, each one potentially producing a new better solution, until the adopted stopping criterion (defined below) is reached.

At each iteration, the algorithm receives as input the current solution, and produces new candidate solutions that are evaluated

[2] By following the nomenclature of the feature selection area (see Kohavi & John, 1997), we referred the implemented techniques as Forward Selection and Backward Elimination.

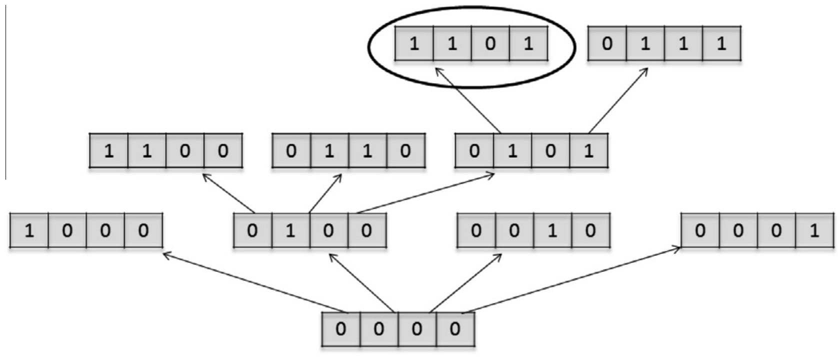


Fig. 1. Search process of *Forward Selection*.

based on the adopted fitness function, also considering the cost constraint. Each candidate solution is produced by including one different test case in the current solution. This way, for a vector solution of size n, each iteration may produce a maximum of n candidate solutions.

The best candidate solution in one iteration becomes the current solution, and is used as input by the next iteration until the overall search process resumes (see Fig. 1).

More formally, new candidate solutions are produced as follows: for each test case t_j not yet present in the current solution $\mathbf{t}$ (i.e., for each $t_j = 0$), a new candidate solution $\mathbf{t}'$ is produced by setting $t_j \leftarrow 1$. For each candidate solution, the fitness function *Fitness*($\mathbf{t}'$) is computed. We also verify whether the candidate solution under analysis is feasible considering the cost constraint (i.e., whether its cost is not higher than the threshold θ). The feasible candidate solution which yields the highest fitness value is then adopted as the new current solution in the search process.

The algorithm stops (1) when no feasible solution is found at an iteration, or (2) when all test cases have been already added to the current solution. An example of this process is shown in Fig. 1.

The FS strategy is simpler to implement and computationally faster than the BCPSO algorithm. In our experiments, this technique obtained better results than the BCPSO in isolation. However, it was overcome by the hybrid BCPSO-FS algorithm (see Sections 3.5 and 4).

The main disadvantage of this technique is that it does not provide a mechanism for excluding a test case in the candidate solution that was added to the solution set at a previous iteration. Note that, in our context, further additions may turn a particular test case unnecessary.[3]

3.4.2. Backward Elimination

Backward Elimination (BE), or Sequential Backward Selection, is the topdown analogy to forward selection (Webb, 2002).

Applied to our context, the BE algorithm starts with a complete solution (i.e., with all $t_j = 1$), and iteratively removes one test case from the current solution until a stopping criterion is reached.

For each test case present in the current solution (i.e., for each $t_j = 1$), a candidate solution $\mathbf{t}'$ is produced by setting $t_j \leftarrow 0$ in $\mathbf{t}$. The fitness function (Eq. (1)) is computed and the candidate solution which yields the highest value of fitness is considered as the current solution for the next iteration.

This process is repeated until the first feasible solution is found. Fig. 2 illustrates this process.

In the performed experiments, both BE and FS delivered better results than the BCPSO in isolation. However, BE is computationally more expensive than FS, since the fitness function must be evaluated over larger sets of test cases (Webb, 2002). Yet, the BE algorithm was also overcome by the hybrid BCPSO-FS (see Section 4).

Finally, note that this algorithm does not aim to improve an already feasible solution. As said, it stops when the first feasible solution regarding cost constraint is generated. This way, it was not worthy to implement a hybrid BCPSO-BE, since the number of infeasible particles during the BCPSO search process is low.

3.4.3. Hill Climbing

The Hill Climbing (referred in this work as HC) is a simple iterative local search algorithm that starts with a random solution to a problem, and progressively tries to find better solutions by using a local search operator. At each iteration, this algorithm generates candidate solutions by performing small changes to the current solution (these solutions are said to be the *neighbors* of the current solution). The best candidate solution of one iteration becomes the new current solution only if it yields a better fitness value than the previous one. This process stops when no better neighbors to the current solution can be found.

The HC algorithm starts with a random feasible solution $\mathbf{t}$ with each t_j randomly chosen (i.e., each t_j receives 0 or 1 value with the same probability). At each iteration, a set of n neighbors $S = s_1, \ldots, s_n$ are generated. Each neighbor is derived from the current solution by inverting the value of a randomly chosen $t_j \in \mathbf{t}$. This operator performs local changes in the current solution $\mathbf{t}$ in order to refine the search in the region being currently explored. The solution $\mathbf{t}$ is updated as:

$$\mathbf{t} = \begin{cases} \hat{s} & \text{if} \quad Fitness(\hat{s}) > Fitness(\mathbf{t}) \\ \mathbf{t} & \text{otherwise} \end{cases} \tag{9}$$

where $\hat{s}$ is given by:

$$\hat{s} = \{max\{Fitness(s_1), \ldots, Fitness(s_n)\} | Cost(s_j) \leqslant \theta\} \tag{10}$$

The algorithm stops when (1) $\hat{s} = \emptyset$ (i.e., no generated neighbor satisfies the cost constraint defined by the user), or (2) $Fitness(\hat{s}) < Fitness(\mathbf{t})$. The search process is then restarted with a new random solution (see Russell & Norvig, 2009 for details). This process is repeated until a maximum number of restarts is performed. The final solution returned by the algorithm is the best solution of all random restarts.

In the performed experiments, the HC was overcome by all other implemented algorithms. Nevertheless, the hybrid BCPSO-HC performed better than the BCPSO in isolation (see Section 4).

3.5. Hybrid algorithms

According to Chen, Qin, Liu, and Lu (2005), hybrid algorithms combining global search with local search (also named Memetic

[3] See (Webb, 2002) for more details about this problem.

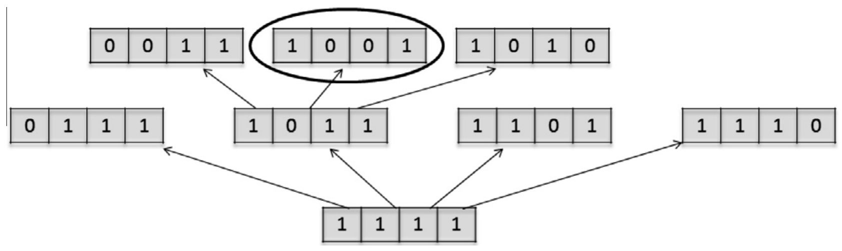


Fig. 2. Search process of *Backward Elimination*.

algorithms) have shown to be very successful in solving several optimization problems.

In this light, we developed two PSO hybrid algorithms aiming to verify whether some improvement in PSO performance could be obtained by using a local search mechanism for each particle. We developed (1) the BCPSO-FS, by combing the BCPSO with the Forward Selection algorithm; and (2) the BCPSO-HC, by using the Hill Climbing algorithm as a local search mechanism.

The BCPSO-FS algorithm is similar to BSPSO, however, at each BCPSO iteration, the FS algorithm is used as a local search mechanism in order to refine each particle (solution). Each particle produced by a BCPSO iteration is given as an initial solution to the FS, which refines it until the FS' stopping criterion is reached. The solutions optimized by FS algorithm are then used as the particles population of the next BCPSO iteration. This way, this hybrid algorithm alternates the use of BCPSO and FS during the search: BCPSO performs a global exploration of the search space, whereas FS refines the solutions provided by BCPSO by performing a local search.

Similarly to the BCPSO-FS algorithm, the BCPSO-HC uses the Hill Climbing algorithm as the local mechanism, trying to refine each particle (i.e., each solution). In this case, we do not use the random restart when performing the HC because our aim here is just to refine the current solution, and not to generate a completely new particle.

Our experiments indicated that the combination of the BCPSO with FS and with HC mechanisms indeed improved the quality of the search results (see Section 4).

As said before, we did not implement a hybrid BCPSO-BE because the BE algorithm can only improve infeasible solutions, which are very reduced in the BCPSO search process.

3.6. Random approach

Finally, as a basis of comparison, we also performed experiments using a *purely random* search algorithm, which, despite its simplicity, has the advantage of performing a uniform exploration of the search space, being very competitive in other contexts of Software Testing (Takaki et al., 2010).

Basically, it operates by generating random solutions **t** and evaluating their fitness values. The algorithm returns the best feasible solution among all generated ones. It is important to highlight that sometimes the random algorithm does not generate any feasible solution after all iterations. In this case, we consider that no solution was returned by this algorithm.

All the algorithms presented in this section were tested using two real world functional test suites. As expected, the Random algorithm was overcome by all other implemented algorithms in this work.

4. Experiments and results

This section presents the experiments performed in order to evaluate the search algorithms implemented in this work. The experiments were performed on a case study related to mobile devices. Other case studies in different domains will be performed as future work.

Table 1
Characteristics of the test suites.

	Integration suite	Regression suite
Total effort to execute all test cases	1053.91 min	699.31 min
# of Requirements	410	248
Redundancy (%)	0.36	14.09
# of Test Cases	80	80

4.1. Experiments preparation

Initially, we selected two test suites related to different features in the context of mobile devices[4]: an Integration Suite and a Regression Suite. Both suites have 80 TCs, each one representing an exhaustive test case scenario (see Table 1). The Integration Suite (which covers 410 requirements) is focused on testing whether the various features of a mobile device can work together, i.e., whether the integration of the features behaves as expected. The Regression Suite (covering 248 requirements), in turn, is aimed at testing whether updates to a specific main feature (e.g., the message feature) have not introduced faults into the already developed (and previously tested) feature functionalities.

Here, test cases are written in a controlled natural language, and contain annotations which allow us to identify which requirements are covered by each test case.

Due to their nature, it is expected that the test suites used in our experiments are different regarding the redundancy in the requirements covered by their test cases. Here, we measure redundancy in a test suite by averaging the Jaccard similarity (Eq. (11)) between the sets of requirements covered by each pair of TCs in the suite. Given two test cases respectively covering the requirements sets A and B, the Jaccard measure is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{11}$$

The above measure was averaged over all pairs of TCs in each suite, in order to indicate the amount of requirements simultaneously covered by different test cases in the suite. The results of this measure are presented in Table 1.

We can observe that the Integration Suite is less redundant (i.e., two distinct test cases rarely cover the same requirements). Hence, for this suite, it is expected to be more difficult to find a solution (subset of test cases) with a good fitness evaluation, since in this case it is more difficult to eliminate a test case without losing coverage.

In the Regression Suite, in turn, each test case individually covers a higher number of requirements. The higher level of redun-

[4] These suites were created by test engineers of the Motorola CIn-BTC (Brazil Test Center) research project.

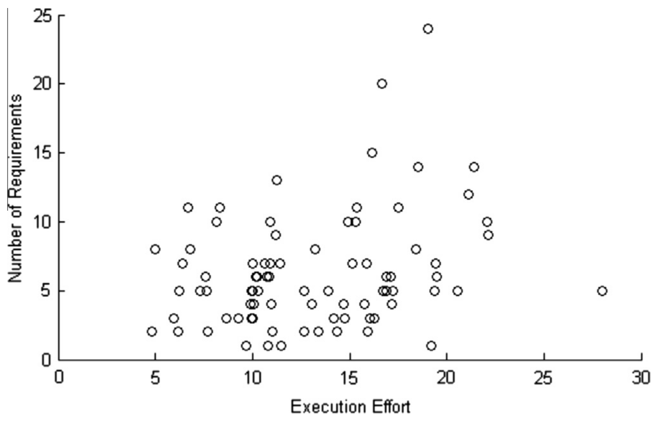


Fig. 3. Integration suite-execution effort × Requirements.

dancy makes it easier to eliminate test cases in the Regression Suite preserving good coverage.

It is important to highlight that the requirements covered by the suites have no overlap (i.e., the requirements covered by the Integration Suite are distinct from the requirements covered by the Regression Suite). Our aim here was to evaluate independent selection scenarios with different internal redundancies to verify the performance of the algorithms.

The used test suites are also different regarding their execution effort. The effort to execute each test case was measured in our work by the Test Execution Effort Estimation Tool, developed by Aranha and Borba (2008). The effort represents the cost (in time) needed to manually execute each test case on a particular mobile device. More details about this tool can be seen in Section 4.1.1. Both suites have 80 test cases, however the Integration Suite is more complex, since the total effort needed to execute its test cases is higher when compared to the total effort associated to Regression Suite (see Table 1).

Finally, Figs. 3 and 4 show the relation between the number of requirements and the execution effort for each test case of each suite. By inspecting these figures, it is not possible to verify a linear relationship between number of requirements and the execution effort. In fact, the correlation coefficient values observed for each suite (0.27 for the Integration Suite and 0.10 for the Regression Suite) are relatively low.

4.1.1. *Test Execution Effort Estimation Tool*

This section briefly presents the estimation model for test execution effort proposed by Aranha and Borba (2007), used in the present work to compute the test suites execution time (which is the constraint in our optimization search process). This model was implemented in the Test Execution Effort Estimation Tool (Aranha & Borba, 2008), and was evaluated through an empirical study on the mobile phone application domain, having obtained higher accuracy than estimation models based on historical test productivity.

In this model, the execution effort of a test suite is computed based on the estimated time spent to manually execute each of its tests. The cost of executing each test is calculated based on its specification -usually, a test specification includes pre-conditions, procedure (steps, inputs and expected outputs) and post-conditions.

This model defines a measure of size and execution complexity of a test case (its costs) in terms of *Execution Points (EP)* associated to each of its step (as presented below). The execution time of an EP (in seconds) is given as an input parameter, and it may be calculated based on historical data of test execution in a particular application domain (see Aranha & Borba, 2008 for details).

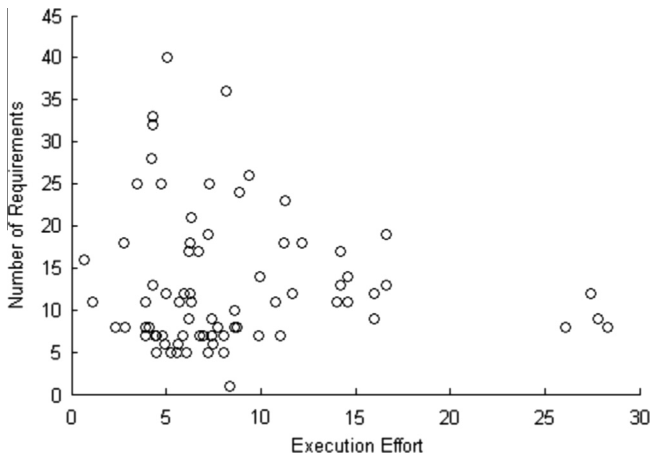


Fig. 4. Regression suite-execution effort × Requirements.

Table 2
Mean fitness, standard deviation and ranking groups for integration suite.

	Mean fitness	Std. Deviation	Ranking group
BCPSO-FS	71.8126	1.9362	1
FS	70.7783	1.3300	2
BCPSO-HC	70.6149	1.3674	2
BE	70.5721	1.1559	2
BCPSO	68.7574	1.5392	3
HC	67.2191	3.6401	4
Random	55.2375	7.1359	5

The test size is given by the total amount of test steps – or *test actions* (each action corresponding to one EP), whereas the test execution complexity is calculated based on the estimated cost of executing each functional (e.g., number of pressed keys and number of screen navigation) and non-functional (e.g., use of network) feature appearing in the test steps.[5] The cost of each feature will correspond to a certain number of EPs, and this amount may be adjusted by experts in testing, according to the test conditions and the application domain (see Aranha & Borba, 2008 for details). Remind that the value of the EP is given as a parameter, and may also be adjusted according to the application domain.

Finally, the cost (in terms of time) of a test suit is obtained by summing up the total amount of EPs of its test cases – which represent the size and execution complexity of the whole test suite.

4.2. *Experiments execution*

Each implemented search technique was executed in each test suite, also varying the execution effort threshold. Each experiment setting was replicated 30 times (in order to allow statistical comparison), thus yielding 7980 executions. The execution settings are seen below:

- 2 test suites: Integration and Regression test suites;
- 7 search techniques: BCPSO, BCPSO-FS, BCPSO-HC, HC, FS, BE and Random;
- 19 execution effort thresholds: varying from 5% to 95% of total effort, with increments of 5%. Each value represents a boundary (limit) which has to be attended by the subsets yielded by each search technique;
- 30 replications.

[5] The test execution complexity reflects the difficulty of interaction of the tester with the product under test during test execution.

Table 3
Mean differences between techniques for integration suite.

Algorithm	- Algorithm	Difference	Std Err Dif	Lower CL	Upper CL	p-Value
BCPSO-FS	Random	16.57510	0.1946748	16.0008	17.14936	<.0001*
FS	Random	15.54086	0.1946748	14.9666	16.11513	<.0001*
BCPSO-HC	Random	15.37741	0.1946748	14.8031	15.95167	<.0001*
BE	Random	15.33462	0.1946748	14.7604	15.90888	<.0001*
BCPSO	Random	13.51990	0.1946748	12.9456	14.09416	<.0001*
HC	Random	11.98160	0.1946748	11.4073	12.55586	<.0001*
BCPSO-FS	HC	4.59350	0.1946748	4.0192	5.16776	<.0001*
FS	HC	3.55926	0.1946748	2.9850	4.13353	<.0001*
BCPSO-HC	HC	3.39581	0.1946748	2.8215	3.97007	<.0001*
BE	HC	3.35302	0.1946748	2.7788	3.92728	<.0001*
BCPSO-FS	BCPSO	3.05520	0.1946748	2.4809	3.62946	<.0001*
FS	BCPSO	2.02097	0.1946748	1.4467	2.59523	<.0001*
BCPSO-HC	BCPSO	1.85751	0.1946748	1.2832	2.43177	<.0001*
BE	BCPSO	1.81472	0.1946748	1.2405	2.38898	<.0001*
BCPSO	HC	1.53830	0.1946748	0.9640	2.11256	<.0001*
BCPSO-FS	BE	1.24048	0.1946748	0.6662	1.81474	<.0001*
BCPSO-FS	BCPSO-HC	1.19769	0.1946748	0.6234	1.77195	<.0001*
BCPSO-FS	FS	1.03423	0.1946748	0.4600	1.60850	<.0001*
FS	BE	0.20625	0.1946748	-0.3680	0.78051	0.9397
FS	BCPSO-HC	0.16346	0.1946748	-0.4108	0.73772	0.9808
BCPSO-HC	BE	0.04279	0.1946748	-0.5315	0.61705	1.0000

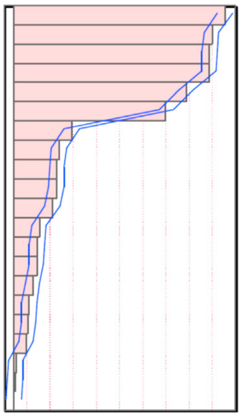


Table 4
Mean fitness, standard deviation and ranking groups for regression suite.

	Mean fitness	Std. deviation	Ranking group
BCPSO-FS	91.7409	4.1195	1
FS	91.4976	3.9220	1
BE	91.0349	3.4438	1
BCPSO-HC	90.7506	3.0293	1
BCPSO	89.2487	2.1378	2
HC	88.5781	4.5878	2
Random	74.3930	18.3534	3

Depending on the search technique, it was necessary to define some additional settings. As suggested in Shi and Eberhart (1998), for PSO based techniques the additional settings were:

- Population size: 20
- Acceleration constants: 1.5
- Topology: Ring (*lbest*)
- Inertia Weight: linearly decreasing from 0.9 to 0.4

The Hill Climbing was implemented with random restart (see Russell & Norvig, 2009). Finally, for all algorithms (when applicable), the stop criterion was the maximum number of fitness evaluations (FEs), here defined as 200,000 FEs. This number showed to be big enough to allow the convergence of the techniques.

4.3. Results

In this section, we start by presenting the results for the Integration Suite, followed by the results achieved for the Regression Suite. We highlight that all presented results were statistically evaluated by using the Tukey HSD multiple comparison test.[6]

4.3.1. Integration Suite Results

Table 2 shows the mean fitness (requirements coverage) and standard deviation of each search technique for the Integration Suite. The ranking groups identified by the Tukey HSD test are shown. The techniques arranged in the same group obtained statistically equivalent results (with a 95% level of confidence).

Additionally, the comparison between each individual pair of techniques can be seen in Table 3. The symbol * along the p-values indicates that the mean difference is significant (with 95% of confidence). Table 3 also presents the 95% confidence intervals of these differences.

It is possible to observe that the BCPSO-FS technique obtained the best average results, outperforming the other techniques. The second ranking group contains the FS, BCPSO-HC and BE algorithms, which were outperformed only by the BCPSO-FS technique. The experiments also revealed the good results achieved by the hybrid techniques compared to their individual components. Both BCPSO-FS and BCPSO-HC were better than BCPSO and than their local search components. Based on our experiments, the hybrid strategies have shown to be very promising to treat the TC selection problem, and thus they should be further researched.

Finally, we highlight that good results were also achieved by the local search techniques FS and BE. These techniques have the advantage of being simpler to implement and less computationally expensive. Hence, they should be indicated when the user needs a result quickly.

4.3.2. Regression Suite Results

Table 4 shows the mean fitness and standard deviation of each search technique for the Regression Suite, as well as the 3 ranking groups derived from the Tukey HSD test. Following, Table 5 shows the mean differences between the search techniques in a pairwise comparison.

Differently from the Integration Suite, it was not possible to identify a single best search technique for the Regression Suite. As seen in Table 4, the BCPSO-FS, FS, BE and BCPSO-HC were considered statistically equivalent (Group 1). The second group was composed by the BCPSO and HC algorithms, followed by the Random technique in Group 3.

Yet, similarly to the results observed in the Integration Suite, the hybrid techniques outperformed their individual components in absolute terms. However, the performance gain was not statistically confirmed in all cases. In this suite, only the hybrid BCPSO-HC was statistically superior to its individual components.

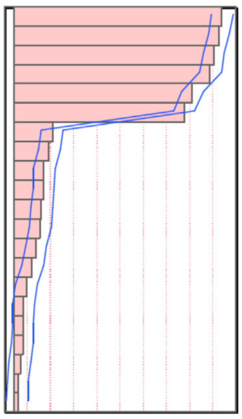
When comparing the obtained results for the Integration and the Regression suites, we observed a significant different level of performance. For the Integration Suite, the average fitness obtained by the algorithms was 67.85 (see Fig. 5). In turn, for the Regression suite, the average fitness was 88.17.

This result confirmed our initial expectations (see Section 4.1). Since the Regression Suite is more redundant regarding requirements coverage than the Integration Suite, the task of selecting TCs from the latter better preserving requirements coverage would be easier.

[6] Using the SAS JMP tool.

Table 5
Mean differences between techniques for regression suite.

Algorithm	- Algorithm	Difference	Std Err Dif	Lower CL	Upper CL	p-Value
BCPSO-FS	Random	17.34791	0.4575965	16.4508	18.24505	<.0001*
FS	Random	17.10456	0.4575965	16.2074	18.00170	<.0001*
BE	Random	16.64191	0.4575965	15.7448	17.53905	<.0001*
BCPSO-HC	Random	16.35753	0.4575965	15.4604	17.25467	<.0001*
BCPSO	Random	14.85569	0.4575965	13.9585	15.75283	<.0001*
HC	Random	14.18506	0.4575965	13.2879	15.08221	<.0001*
BCPSO-FS	HC	3.16285	0.4575965	2.2657	4.05999	<.0001*
FS	HC	2.91950	0.4575965	2.0223	3.81664	<.0001*
BCPSO-FS	BCPSO	2.49222	0.4575965	1.5951	3.38937	<.0001*
BE	HC	2.45685	0.4575965	1.5597	3.35399	<.0001*
FS	BCPSO	2.24887	0.4575965	1.3517	3.14601	<.0001*
BCPSO-HC	HC	2.17247	0.4575965	1.2753	3.06961	<.0001*
BE	BCPSO	1.78622	0.4575965	0.8891	2.68337	<.0001*
BCPSO-HC	BCPSO	1.50184	0.4575965	0.6047	2.39899	0.0010*
BCPSO-FS	BCPSO-HC	0.99038	0.4575965	0.0932	1.88753	0.0305*
FS	BCPSO-HC	0.74703	0.4575965	-0.1501	1.64418	0.1027
BCPSO-FS	BE	0.70600	0.4575965	-0.1911	1.60315	0.1229
BCPSO	HC	0.67063	0.4575965	-0.2265	1.56777	0.1429
FS	BE	0.46265	0.4575965	-0.4345	1.35980	0.3121
BE	BCPSO-HC	0.28438	0.4575965	-0.6128	1.18153	0.5343
BCPSO-FS	FS	0.24335	0.4575965	-0.6538	1.14050	0.5949



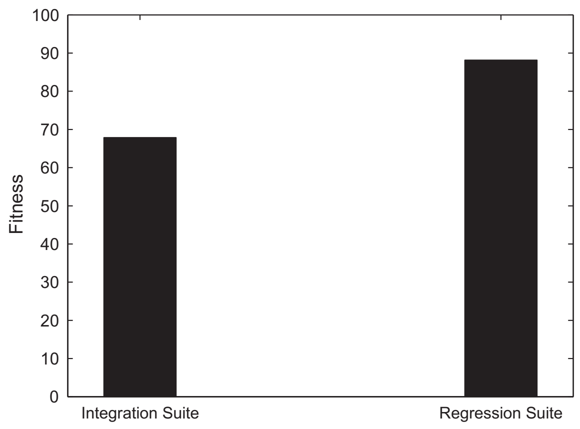


Fig. 5. Average fitness over all executions for each test suite.

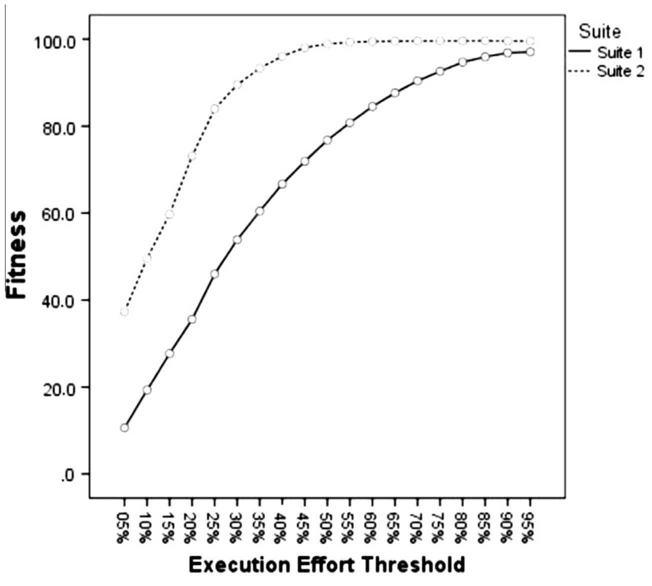


Fig. 6. Fitness by execution effort threshold per Suite.

In fact, for redundant suites, a greater variety of search techniques can be used with good results. For a more complex suite, however, a careful decision should be made regarding which algorithms to adopt for TC selection.

Finally, Fig. 6 presents the average fitness over the search techniques considering different values of effort threshold. As expected, the chosen value of effort threshold has an impact in the general quality of the subsets of test cases returned by the search techniques. When the effort threshold is too low (thus imposing a stronger constraint), the space of feasible solutions is reduced and, hence, the optimization task becomes more difficult.

However, the quality of the solution is also dependent on the complexity of test suite at hand. For the Regression Suite, high values of fitness (greater than 90%) can be actually obtained by adopting relatively low effort limits (from 40% to 50%). For the Integration Suite, in turn, good solutions in terms of coverage can only be obtained using more relaxed constraints.

5. Conclusion

In this work, we investigated the use of hybrid search techniques for the constrained TC selection problem. We can point out some contributions of the current work. First of all, to the best of our knowledge, PSO was not yet investigated in the context of TC selection. Hybrid search techniques were developed and validated in our work. 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 two hybrid algorithms which integrated local search techniques (FS and HC Climbing) to improve the performance of the BCPSO. In the experiments performed using two different test suites related to mobile devices, the hybrid techniques obtained better results when compared to their individual components.

In our experiments, the quality of the test case selection process depended upon: (1) the effectiveness of the used search technique, (2) the constraints imposed by the user (i.e., the effort threshold), and (3) the features (e.g., redundancy) of the test suites at hand.

We observed that a more complex (less redundant) suite required the use of more powerful search techniques in order to achieve satisfactory results in statistical terms. For the simpler suite, a greater variety of algorithms were found to be statistically equivalent in the test case selection task and, in this case, simpler algorithms would be more indicated. The execution effort constraint imposed by the user also influenced the quality of the returned solutions, but the observed impact varied depending on the features of the test suites.

The benefits of using more complex and effective search techniques will vary by organization. However, even when small, an increase in requirements coverage may be critical in some contexts (e.g., large test suites and/or strong resources constraints). Yet, the effort required to use the proposed algorithms/techniques is not so high, since the complexity of PSO and the hybrid techniques (when compared to simpler search techniques) is transparent to the final user (the software engineer responsible for the automatic selection process). Hence, the increase in fitness can be obtained without extra human effort. Note that the only process that could demand human attention would be the design (values assignment) of parameters. However, in our work we observed good results by adopting the PSO default parameters values suggested in the literature (i.e., no parameter optimization was performed).

The current research provides a framework for future developments that are intended to progressively improve the quality of the selection process. Several extensions of the current work can be considered in the near future. First, we intend to perform experiments on more test suites. Second, our work was focused on a constrained formulation of TC selection using only one objective function (requirements coverage) in the optimization process. However, other criteria can be deployed, which will require the use of constrained multi-objective techniques. Finally, we will investigate new strategies to combine search techniques, in order to provide more robust hybrid algorithms for TC selection. In fact, the approach of alternating global and local search adopted in our work is only one specific strategy to produce hybrid search algorithms.

Acknowledgements

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES www.ines.org.br), CNPq, CAPES and FACEPE.

References

Aranha, E., & Borba, P. (2007). An estimation model for test execution effort. In *Proceedings of the 1st international symposium on empirical software engineering and measurement* (pp. 107–116).

Aranha, E., & Borba, P. (2008). Using process simulation to assess the test design effort reduction of a model-based testing approach. In ICSP, (pp. 282–293).

Barltrop, K., Clement, B., Horvath, G., & Lee, C.-Y. (2010). Automated test case selection for flight systems using genetic algorithms. In *Proceedings of the AIAA Infotech@Aerospace conference (I@A 2010) Atlanta, GA*.

Beizer, B. (1990). *Software testing techniques*. International Thomson Computer Press.

Borba, P., Torres, D., Marques, R., & Wetzel, L. (2007). Target – test and requirements generation tool. In *Motorola's 2007 innovation conference (IC'2007)*.

Borba, P., Cavalcanti, A., Sampaio, A., & Woodcock, J. (2007). Testing techniques in software engineering, Second Pernambuco summer school on software engineering, PSSE 2007, Recife, Brazil, December 3–7, Revised Lectures, Lecture notes in computer science, Springer, Vol. 6153, 2010.

Cartaxo, G. E., Machado, D. L. P., & Oliveira Neto, G. F. (2009). On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability, 21*(2), 270–285.

Chen, J., Qin, Z., Liu, Y., & Lu, J. (2005). Particle swarm optimization with local search. In *International conference on neural networks and brain, ICNN B '05* (Vol. 1, pp. 481–484).

Chen, T. Y., & Lau, M. F. (1998). A new heuristic for test suite reduction. *Information & Software Technology, 40*(5–6), 347–354.

Eberhart, R. C., & Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. *LNCS, 1447*, 611–616.

Elbaum, S., Malishevsky, A., & Rothermel, G. (2001). Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd international conference on software engineering, ICSE '01* (pp. 329–338). Washington, DC, USA: IEEE Computer Society.

Harold, M. J., Gupta, R., & Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering Methodology, 2*(3), 270–285.

Hodgson, R. J. W. (2002). Partical swarm optimization applied to the atomic cluster optimization problem. In *Proceedings of the genetic and evolutionary computation conference, GECCO '02* (pp. 68–73). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hu, X., & Eberhart, R. (2002). Solving constrained nonlinear optimization problems with particle swarm optimization. In *6th world multiconference on systemics, cybernetics and informatics* (pp. 203–206).

Juang, C.-F. F. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, 34*(2), 997–1006.

Kaur, A., & Bhatt, D. (2011). Hybrid particle swarm optimization for regression testing. *International Journal on Computer Science and Engineering, 3*(5), 1815–1824.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE international joint conference on neural networks* (pp. 1942–1948).

Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *Proceedings of the world multiconference on systemics, cybernetics and informatics* (pp. 4104–4109).

Kissoum, Y., & Sahnoun, Z. (2007). A formal approach for functional and structural test case generation in multi-agent systems. In *IEEE/ACS international conference on computer systems and applications* (pp. 76–83).

Kohavi, R., & John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*(1-2), 273–324.

Lin, J.-W., & Huang, C.-Y. (2009). Analysis of test suite reduction with enhanced tie-breaking techniques. *Information and Software Technology, 51*(4), 679–690.

Lvbjerg, M., Rasmussen, T. K., & Krink, T. (2001). Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 469–476). Morgan Kaufmann.

Malishevsky, A. G., Ruthruff, J. R., Rothermel, G., & Elbaum, S. (2006). Cost-cognizant test case prioritization, Tech. rep., Department of Computer Science and Engineering, University of Nebraska-Lincoln.

Mansour, N., & El-Fakih, K. (1999). Simulated annealing and genetic algorithms for optimal regression testing. *Journal of Software Maintenance, 11*(1), 19–34.

Ma, X.-Y., Sheng, B.-K., & Ye, C.-Q. (2005). Test-suite reduction using genetic algorithm. *Lecture Notes in Computer Science, 3756*, 253–262.

Ramler, R., & Wolfmaier, K. (2006). Economic perspectives in test automation – balancing automated and manual testing with opportunity cost. In *Workshop on automation of software test, ICSE 2006*.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In *Proceedings of the 7th international conference on evolutionary programming* (pp. 591–600).

Souza, L. S., Prudencio, R. B. C., & Barros, F. D. A. (2010) A constrained particle swarm optimization approach for test case selection. In *Proceedings of the 22nd international conference on software engineering and knowledge engineering (SEKE 2010) Redwood City, CA, USA*.

Souza, L. S., Miranda, P. B. C., Prudencio, R. B. C., & Barros, F. D. A. (2011). A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In *Proceedings of the 23rd international conference on tools with artificial intelligence (ICTAI 2011) Boca Raton, FL, USA*.

Takaki, M., Cavalcanti, D., Gheyi, R., Iyoda, J., d'Amorim, M., & Prudêncio, R. B. (2010). Randomized constraint solvers: A comparative study. *Innovations in Systems and Software Engineering: A NASA Journal, 6*(3), 243–253.

Tukey, J. W. (1949). Comparing Individual Means in the Analysis of Variance. *Biometrics, 5*(2), 99–114.

Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., & Roos, R. S. (2006). Timeaware test suite prioritization. In *Proceedings of the 2006 international symposium on software testing and analysis* (pp. 1–12).

Webb, A. R. (2002). *Statistical pattern recognition* (2nd ed.). John Wiley & Sons.

Windisch, A., Wappler, S., & Wegener, J. (2007). Applying particle swarm optimization to software testing. In *Proceedings of the 9th annual conference on genetic and evolutionary computation, GECCO'07* (pp. 1121–1128). New York, NY, USA: ACM.

Yoo, S., & Harman, M. (2007). Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on software testing and analysis* (pp. 140–150).

Yoo, S., & Harman, M. (2010). Using hybrid algorithm for parento efficient multi-objective test suite minimisation. *Journal of Systems and Software, 83*, 689–701.

Yoo, S., & Harman, M. (2010). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability, 22*(2), 67–120.