A Multi-Objective Particle Swarm Optimization for Test Case Selection Based on Functional Requirements Coverage and Execution Effort

Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio, Flavia de A. Barros

Center of Informatics (CIn), Federal University of Pernambuco (UFPE)
Recife (PE) - Brazil
{lss2, pbcm, rbcp, fab}@cin.ufpe.br

Abstract—Although software testing is a central task in the software lifecycle, it is sometimes neglected due to its high costs. Tools to automate the testing process minor its costs, however they generate large test suites with redundant Test Cases (TC). Automatic TC Selection aims to reduce a test suite based on some selection criterion. This process can be treated as an optimization problem, aiming to find a subset of TCs which optimizes one or more objective functions (i.e., selection criteria). The majority of search-based works focus on single-objective selection. In this light, we developed a mechanism for functional TC selection which considers two objectives simultaneously: maximize requirements’ coverage while minimizing cost in terms of TC execution effort. This mechanism was implemented as a multi-objective optimization process based on Particle Swarm Optimization (PSO). We implemented two multi-objective versions of PSO (BMOPSO and BMOPSO-CDR). The experiments were performed on two real test suites, revealing very satisfactory results (attesting the feasibility of the proposed approach). We highlight that execution effort is an important aspect in the testing process, and it has not been used in a multi-objective way together with requirements coverage for functional TC selection.

Keywords—Software testing; Test case selection; Multi-objective optimization; PSO; Particle Swarm Optimization.

I. INTRODUCTION

In a competitive world, where custumers search for high quality products, the software testing activity has grown in importance, aiming to assure quality and reliability to the product under development. Nevertheless, this activity is sometimes neglected, since it is very expensive, reaching up to 40% of the final software development cost [1]. Thus, it is of central importance to improve the efficiency and effectiveness of the whole testing process.

We can identify in the related literature two main approaches for software testing: structural (white box) testing, which investigates the behavior of the software directly accessing its code; and functional (black box) testing, which investigates whether the software functionalities are responding/behaving as expected [2]. In both approaches, software testing is conducted via the execution of a Test Suite, that is, a set of Test Cases. A Test Case (TC), in turn, consists of “a set of inputs, execution conditions, and a pass/fail condition” [2].

Before starting the testing process, it is necessary to choose a test adequacy criterion in order to evaluate whether a suite is adequate with respect to the test goals [3]. A test adequacy criterion defines properties that must be observed by the test suite (e.g., code coverage, functional requirements’ coverage, among others).

Several tools have been proposed to automate or optimize software testing tasks, from test generation to its execution. Regarding automatic TC generation, we can identify tools which generate test suites from some software artifact (such as code, functional requirements, use cases) [4], [5]. However, as these tools generate TCs in a systematic way (aiming to provide a good coverage of the testing adequacy criterion), the generated test suites are usually too large to be executed with the available resources (tools, time, people) [6].

When analyzing large test suits, we can identify redundancies in TCs. Hence, it is possible to cut down these suits, in order to fit the available resources, without severely compromising the coverage of the test adequacy criterion being observed. The task of reducing a test suite based on some selection criterion is known as “Test Case selection” [7]. The test selection criterion depends on the test adequacy criterion being used.

Clearly, TC selection should not be performed at random, in order to preserve the coverage of the testing criterion. In the absence of automatic tools, this task is usually manually performed in an ad-hoc fashion. However, manual TC selection is time-consuming and susceptible to errors.

Different authors try to automatically solve the TC selection problem by deploying a variety of techniques. Some works focus on deterministic software engineering solutions (see [7]). Despite their good results, these works consider only a single criterion for TC selection. Yet, they are computationally expensive when dealing with large test suites.

On the other hand, some authors deploy artificial intelligence search-based techniques to find a subset of TCs which optimizes a given objective function (i.e., a given selection criterion). Yet, the majority of search-based works focus on single-objective selection (e.g., [6]’s work using Greedy Search for structural TC selection; and [8]’s work using PSO for functional TC selection). Regarding multi-
objective selection, we cite Yoo and Harman’s [9] work, which uses Genetic Algorithms for structural TC selection.

In this light, we developed a mechanism for functional TC selection which considers two objectives (selection criteria) simultaneously: maximize requirements’ coverage (quality of the test suite) while minimizing cost in terms of execution effort (i.e., the time necessary to execute the TC).

This work investigated the use of Particle Swarm Optimization (PSO) [10] for multi-objective TC selection. 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 [19]. We implemented two different algorithms: a Binary Multi-Objective PSO (BMOPSO) [11], [12]; and a modified version, known as BMOPSO-CDR (which uses Crowding Distance and Roulette Wheel) [13]. These algorithms provide to the user a set of solutions (test suites) with different values of requirements’ coverage versus execution effort. The user can then choose the solution that best fits the available resources for executing the tests. The implemented algorithms were executed on two different real-world test suites of functional TCs related to the mobile devices domain. They were evaluated using five well-known multi-objective metrics. The performed experiments obtained very satisfactory results.

We did not find in the related literature any work focusing on multi-objective functional TC selection. Besides, we highlight that, although TC execution effort (cost) is a very important aspect in the testing process, it has been neglected by the search-based previous work. In fact, most of the related work considers only requirements or code coverage.

Following, section 2 briefly discusses the Search-Based TC selection approach. Section 3 presents the proposed algorithms. Section 4 brings experiments and obtained results and Section 5 presents conclusions and future work.

II. SEARCH-BASED TEST CASE SELECTION

According to Lin [6] Test Case selection can be treated as an optimization problem in which search techniques explore the space of possible solutions (subsets of TCs) seeking the solution that best matches the test objectives. Nevertheless, the returned subset may not have the same coverage of the original test suite with respect to the test adequacy criterion being used.

Deterministic approaches do not seem suitable for TC selection, since this is known to be an NP-Complete problem [6]. Thus, heuristic search approaches appear to be more feasible to treat this problem. Within this approach, we cite the HGS algorithm [14], the GRE algorithm [15], Greedy Search heuristics for set-covering [16], Genetic Algorithms [9] and PSO [8].

Besides the technique to be used in TC selection, another central issue is the test selection criterion. For structural testing, the most commonly used selection criterion consists of the amount of “pieces” of program code (e.g., blocks, statements, decisions) that should be exercised by TCs [14]. In turn, for functional testing, the most usual criterion is the amount of functional requirements covered by the selected TCs [7], [8].

In fact, in the related literature we identified few works which take into account the cost (effort) to execute the test cases [9], [17], [8]. For functional tests, in particular, it is difficult to consider the TC execution cost because, for that, it is necessary to estimate the cost of manually performing the TC [18].

Finally, we see that the majority of existing works in TC selection considers only one selection criterion at a time, using some single-objective method to optimize the chosen objective. However, for TC selection in general, it is desirable to consider more than one criterion at the same time (e.g., maximum requirements’ coverage and minimum execution effort cost).

For that, it is necessary to work within the multi-objective approach, in which the search heuristic tries to optimize different objectives at the same time. The following section brings some concepts of multi-objective optimization, providing a better understanding of the work presented in this paper.

III. MULTI-OBJECTIVE PSO TO TEST CASE SELECTION

In this work, we propose a method that adopts Particle Swarm Optimization (PSO) to solve multi-objective TC selection problems. In contrast to single-objective problems, Multi-Objective Optimization (MOO) aims to optimize more than one objective at the same time.

A MOO problem considers a set of k objective functions \( f_1(x), f_2(x), \ldots, f_k(x) \) in which \( x \) is an individual solution for the problem being solved. The output of a MOO algorithm is usually a population of non-dominated solutions considering the objective functions. Formally, let \( x \) and \( x' \) be two different solutions. We say that \( x \) dominates \( x' \) (denoted by \( x \preceq x' \)) if \( x \) is better than \( x' \) in at least one objective function and \( x \) is not worse than \( x' \) in any objective function. \( x \) is said to be not dominated if there is no other solution \( x_i \) in the current population, such that \( x_i \preceq x \). The set of non-dominated solutions in the objective space returned by a MOO algorithm is known as Pareto front.

We adopted in our work the PSO, which is a population-based search approach, inspired by the behavior of birds flocks [10]. PSO has shown to be a simple and efficient algorithm compared to other search techniques, including for instance the widespread Genetic Algorithms [19]. The basic PSO algorithm starts its search process with a random population (also called swarm) of particles. Each particle represents a candidate solution for the problem being solved and it has four main attributes:

1) the position \( (t) \) in the search space (each position represents an individual solution for the optimization
problem);
2) the current velocity (v), indicating a direction of movement in the search space;
3) the best position (i) found by the particle (the cognitive component);
4) the best position (g) found by the particle’s neighborhood.

For a number of iterations, the particles fly through the search space, being influenced by their own experience (particle memory) and by the experience of their neighbors (social guide). Particles change position and velocity continuously, aiming to reach better positions and to improve the objective functions considered.

We developed in our work a Binary Multi-Objective PSO (BMOPSO) by merging two versions of PSO: (1) the binary version of PSO proposed in [11], since the solutions of the TC selection problem are represented as binary vectors; and (2) the MOPO algorithm originally proposed in [12] to deal with multi-objective problems. Furthermore, we implemented the BMOPSO-CDR algorithm by adding the Crowding Distance and Roulette Wheel mechanism (as proposed by [13]) to the BMOPSO. The proposed algorithms were applied to select functional tests.

The objective functions to be optimized in our work are the functional requirements coverage and the execution effort of the selected TCs, in such a way that we maximize the first function and minimize the second one. The proposed methods return a set of non-dominated solutions (a Pareto front) considering the aforementioned objectives. Figure 1 illustrates a sample of Pareto front return by the proposed algorithms. Each dot in this figure represents the values of the objective functions for a different non-dominated solution returned in the MOO process. By receiving a set of diverse solutions, the user can choose the best one taking into account its current goals and available resources (e.g., the amount of time available at the moment to execute the test cases).

We can point out some previous work that applied PSO in Software Testing, particularly for test case generation [20] and for regression testing prioritization [21]. In our previous work [8], we applied PSO to solve a constrained TC selection problem. However, to the best of our knowledge, there is no previous work investigating PSO on a multi-objective functional TC selection problem. We also highlight that no previous work was found in the literature that performed test selection considering both requirements coverage and execution effort in a multi-objective way in the context of functional software testing.

A. Problem Formulation

In this work, the particle’s positions were defined as binary vectors representing candidate subsets of TCs to be applied in the software testing process. Let \( T = \{ T_1, \ldots, T_n \} \) be a test suite with \( n \) test cases. A particle’s position is defined as \( t = (t_1, \ldots, t_n) \), in which \( t_j \in \{0, 1\} \) indicates the presence (1) or absence (0) of the test case \( T_j \) within the subset of selected TCs.

As said, two objective functions were adopted. The requirements coverage objective (to be maximized) represents the amount (in percentage) of requirements covered by a solution \( t \) in comparison to the amount of requirements present in \( T \). Formally, let \( R = \{ R_1, \ldots, R_k \} \) be a given set of \( k \) requirements of the original suite \( T \). 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 requirements coverage of a solution \( t \) is given as:

\[
R_{Coverage}(t) = 100 \times \frac{\left| \bigcup_{j=1}^{n} \{ F(T_j) \} \right|}{k} \tag{1}
\]

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

The other objective function (to be minimized) is the execution effort (the amount of time required to manually execute the selected suite). Formally, each test case \( T_j \in T \) has a cost score \( c_j \). The total cost (effort) of a solution \( t \) is then defined as:

\[
Cost(t) = \sum_{t_j=1}^{n} c_j \tag{2}
\]

In this approach, the cost \( c_j \) is computed for each test case by using the execution effort estimation model developed by Aranha and Borba [18]. The MOO algorithms will be used to find a good Pareto front regarding the objective functions \( R_{Coverage} \) and \( Cost \).

B. The BMOPSO Algorithm

As said, the BMOPSO algorithm was created by merging the Multi-Objective Particle Swarm Optimization proposed

The MOPSO uses an External Archive (EA) to store the non-dominated solutions found by the particles during the search process. The EA can be seen as a secondary swarm, which interacts with the primary swarm in order to define the search process. The EA can be seen as a secondary swarm, non-dominated solutions found by the particles during the search process. The EA can be seen as a secondary swarm, non-dominated solutions found by the particles during the search process. The EA can be seen as a secondary swarm, non-dominated solutions found by the particles during the search process. The EA can be seen as a secondary swarm, non-dominated solutions found by the particles during the search process.

The following algorithm summarizes the BMOPSO:

1) Initialize the swarm by randomly initializing the position \( t \) and the velocity \( v \) of each particle;
2) Evaluate each particle according to the considered objective functions and store in the EA the particles’ positions that are non-dominated solutions;
3) Generate hypercubes on the EA objectives’ space and locate each solution in the EA within those hypercubes;
4) Initialize the memory \( \hat{t} \) of each particle as:
\[
\hat{t} = t
\]  
(3)
5) WHILE stop criterion is not verified DO
   a) Compute the velocity \( v \) of each particle by:
\[
v \leftarrow \omega v + C_1 r_1 (\hat{t} - t) + C_2 r_2 (\hat{g} - t)
\]  
(4)
where \( \omega \) represents the inertia factor; \( r_1 \) and \( r_2 \) are random values in the interval \([0,1]\); \( C_1 \) and \( C_2 \) are constants which control the trade-off between the particle’s memory and the social guidance (the social guide \( g \) in the above equation is defined as one of the non-dominated solutions stored in the current EA. The following procedure is adopted in order to select the \( g \) component for each particle:
   i) Select one hypercube of the EA by using Roulette Wheel, in which the probability of choosing a given hypercube is proportional to the number of solutions located in the hypercube. Hence, the hypercubes with less solutions (thus, corresponding to not well explored regions in the objectives’ space) has a higher probability of being selected;
   ii) Select a random solution within such hypercube as the \( g \) value;
   b) Compute the new position \( t \) of each particle as:
\[
t = \begin{cases} 
1, & \text{if } r_3 \leq \text{sig}(v) \\
0, & \text{otherwise}
\end{cases}
\]  
(5)
\[\text{where } r_3 \text{ is a random number sampled in the interval } [0,1]\text{ and } \text{sig}(v) \text{ is defined as:}
\[
\text{sig}(v) = \frac{1}{1 + e^{-v}}
\]  
(6)
Equations (5) and (6) were proposed in the original binary PSO [11] in order to certify that the new positions are still binary vectors. The position value \( t \) tends to 0 when the velocity assumes higher values. In turn, \( t \) tends to 0 for lower values of velocity;
   c) Use the mutation operator as proposed by [12];
   d) Evaluate each particle of the swarm and update the solutions stored in the EA by inserting all the new non-dominated solutions and by removing all previous EA’s solutions that were dominated. Since the size of the EA is limited, whenever it gets full, the solutions in more populated hypercubes are removed from the EA;
   e) Update the particle’s memory \( \hat{t} \) using eq. (3) if the new position \( t \) dominates the previous memory (i.e., if \( t \preceq \hat{t} \)). If neither \( t \preceq \hat{t} \) nor \( \hat{t} \preceq t \) is verified (i.e., no solution dominates the other one), then the particle’s memory is changed to \( t \) with 50% of probability;
6) END WHILE and return the current EA as the Pareto front.

C. BMOPSO using Crowding Distance and Roulette Wheel

MOPSO using Crowding Distance and Roulette Wheel (MOPSO-CDR) was originally proposed by Santana et al [13] inspired by the MOPSO-CDLS algorithm, proposed by Tsou et al [22].

Instead of using hypercubes to choose the \( \hat{g} \) component, the MOPSO-CDR uses the crowding distance concept (see [23]) on EA’s solutions. Before each iteration, the EA’s solutions are ordered according to its crowding distance values and a roulette wheel is used to select the social guide \( g \) for each particle.

Besides, in order to update the particle’s memory \( \hat{t} \) when neither \( t \preceq \hat{t} \) nor \( \hat{t} \preceq t \), the algorithm searches for EA’s solutions with minimum Euclidean distance to \( t \) and \( \hat{t} \). If the closest EA’s solution to \( t \) has a value of crowding distance lower than the closest EA’s solution to \( \hat{t} \) (e.g., it is in a less crowded region), then eq. 3 is used. Otherwise, \( \hat{t} \) remains the same.

In order to develop the BMOPSO-CDR, we added the CDR mechanism to the BMOPSO previously created.

IV. EXPERIMENTS AND RESULTS

This section presents the experiments performed in order to evaluate the algorithms implemented in our work. Two test
suites, referred here as Suite 1 and Suite 2, from the context of mobile devices\(^2\) were used on the performed experiments. Although both suites have the same size (80 TCs), it is possible to observe in Table I that Suite 1 covers a higher number of requirements than Suite 2, and it is more complex since the total effort needed to execute its test cases is higher when compared to Suite 2.

The TCs in Suite 1 are less redundant\(^3\), i.e. two distinct test cases rarely cover the same requirement. Hence, for Suite 1, it is expected to be more difficult to find non-dominated solutions. 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 eliminate test cases in Suite 2.

### A. Metrics

In our experiments, we evaluated the results (i.e., the Pareto fronts) obtained by the algorithms MOPSO and MOPSO-CDR for each test suite according to five different quality metrics usually adopted in the literature of multi-objective optimization. The following metrics were adopted in this paper: Hypervolume, Spacing, Maximum Spread, Coverage and Coverage Difference. Each metric considers a different aspect of the Pareto front, as follows.

1) **Hypervolume (HV):** this metric is defined by the hypervolume in the space of objectives covered by the obtained Pareto front [24]. For MOO with \(k\) objectives, \(HV\) is defined by:

\[
HV = \left\{ \bigcup_i a_i \mid x_i \in \mathcal{P} \right\},
\]

where \(x_i (i = 1, 2, ..., n)\) are the non-dominated solutions in the Pareto front \((\mathcal{P})\), \(n\) is the number of solutions in \(\mathcal{P}\) and \(a_i\) is the hypervolume of the hypercube delimited by the position of solution \(x_i\) in the space of objectives and the origin.

In practice, this metric computes the size of the dominated space, which is also called the *area under the curve*. A high value of hypervolume is desired in MOO problems.

<table>
<thead>
<tr>
<th>(\text{Suite 1})</th>
<th>(\text{Suite 2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Effort</td>
<td>1053.91 min</td>
</tr>
<tr>
<td># of Requirements</td>
<td>440</td>
</tr>
<tr>
<td>Redundancy</td>
<td>0.36%</td>
</tr>
<tr>
<td># of Test Cases</td>
<td>80</td>
</tr>
</tbody>
</table>

2) **Spacing (S):** this metric estimates the diversity of solutions in the obtained Pareto front [25]. \(S\) is derived by computing the relative distance between adjacent solutions of the Pareto front as follows:

\[
S = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (d_i - \bar{d})^2},
\]

where \(d_i\) is the distance between adjacent solutions to the solution \(x_i\) and \(\bar{d}\) is the average distance between the adjacent solutions. \(S = 0\) means that all solutions of the Pareto front are equally spaced. Hence, values of \(S\) near zero are preferred.

3) **Maximum Spread (MS):** this metric evaluates the maximum extension covered by the non-dominated solutions in the Pareto front [26]. \(MS\) is computed by eq. (9):

\[
MS = \sqrt{\sum_{m=1}^{k} (\max_{i=1}^{n} f_i^m - \min_{i=1}^{n} f_i^m)^2},
\]

where \(k\) is the number of objectives and \(f_i^m\) is the value of the \(m\)-th objective function for the \(i\)-th solution in the Pareto front. High values of this metric are preferred.

4) **Coverage (C):** this is a measure to pairwise comparison of different algorithms [27]. Let \(\mathcal{P}^A\) and \(\mathcal{P}^B\) be the Pareto fronts generated respectively by two techniques \(A\) and \(B\). \(C\) is computed by eq. (10):

\[
C(\mathcal{P}^A, \mathcal{P}^B) = \frac{|\{x' \in \mathcal{P}^B; \exists x \in \mathcal{P}^A : x \preceq x'\}|}{|\mathcal{P}^B|},
\]

where \(\mathcal{P}^A\) and \(\mathcal{P}^B\) are two sets of non-dominated solutions. The value \(C(\mathcal{P}^A, \mathcal{P}^B) = 1\) means that all solutions in \(\mathcal{P}^B\) are dominated by \(\mathcal{P}^A\). On the other hand, \(C(\mathcal{P}^A, \mathcal{P}^B) = 0\) means that none of the solutions in \(\mathcal{P}^B\) is dominated by \(\mathcal{P}^A\). The Coverage metric indicates the amount of the solutions within the non-dominated set of the first algorithm which dominate the solutions within the non-dominated set of the second algorithm.

It is important to highlight that both \(C(\mathcal{P}^A, \mathcal{P}^B)\) and \(C(\mathcal{P}^B, \mathcal{P}^A)\) must be evaluated, since \(C(\mathcal{P}^A, \mathcal{P}^B)\) is not necessarily equal to \(1 - C(\mathcal{P}^B, \mathcal{P}^A)\). If \(0 < C(\mathcal{P}^A, \mathcal{P}^B) < 1\) and \(0 < C(\mathcal{P}^B, \mathcal{P}^A) < 1\), then neither \(\mathcal{P}^A\) totally dominates \(\mathcal{P}^B\) nor \(\mathcal{P}^B\) totally dominates \(\mathcal{P}^A\).

5) **Coverage Difference (D):** Let \(\mathcal{P}^A\) and \(\mathcal{P}^B\) be two sets of non-dominated solutions. The metric \(D\), also proposed by Zitzler [27], is defined by:

\[
D(\mathcal{P}^A, \mathcal{P}^B) = HV(\mathcal{P}^A + \mathcal{P}^B) - HV(\mathcal{P}^B)
\]

This metric gives the size of the space dominated by \(\mathcal{P}^A\) but not dominated by \(\mathcal{P}^B\) (regarding the objective space). In addition, the \(D\) measure gives information about which

---

\(^2\)These suites were created at the Motorola Cln-BTC (Brazil Test Center) research project

\(^3\)Redundancy of a suite is measured by the average Jaccard similarity between the sets of requirements covered by each pair of TCs.
set entirely dominates the other set, e.g., $D(\mathcal{P}^A, \mathcal{P}^B) = 0$ and $D(\mathcal{P}^B, \mathcal{P}^A) > 0$ means that $\mathcal{P}^A$ is dominated by $\mathcal{P}^B$.

B. Algorithms Settings

In this section, we present the values of the parameters adopted for the MOPSO and the MOPSO-CDR algorithms. We adopted the same values suggested by Coello et al [12] and by Santana et al [13]:

- number of particles: 20
- mutation rate: 0.5
- inertia factor $\omega$: linearly decreases from 0.9 to 0.4
- constants $C_1$ and $C_2$: 1.49
- EA's size: 200 solutions
- maximum fitness evaluations: 200,000

We also performed experiments using a purely random search method, as a basis of comparison. Despite its simplicity, the random search has the advantage of performing a uniform exploration of the search space, being very competitive in other software testing contexts (e.g., for test case generation [28]).

C. Results

For statistical analysis purposes, we executed each search algorithm 60 times on each test suite. In each execution, a Pareto front was produced. The values of coverage and cost observed in the Pareto fronts were normalized since they are measured using different scales. After normalizing the Pareto fronts, all the evaluation metrics were computed.

We initially present the results for metrics Hypervolume, Maximum Spread and Spacing, which are computed for each algorithm individually. Tables II and IV show the mean and standard deviation of the metrics Hypervolume, Maximum Spread and Spacing for each algorithm respectively on Suite 1 and Suite 2. Tables III and V, in turn, present the average difference among each pair of algorithms considering these metrics. Asterisk (*) indicates when the differences are statistically significant using the Tukey-Kramer HSD test with a 0.05 level of significance.

From Table II, we observe that the BMOPSO-CDR outperformed both BMOPSO and Random algorithms in terms of Hypervolume, Maximum Spread and Spacing for Suite 1. The gain obtained by the BMOPSO-CDR has shown to be statistically significant for all evaluated metrics (see Table III). For Suite 1, the BMOPSO-CDR covered a wider region of the search space and also generated a larger Pareto front with better spaced solutions. BMOPSO also statistically outperformed the Random algorithm in terms of Hypervolume and Maximum Spread, but it was equivalent to it considering the Spacing metric.

For Suite 2, the BMOPSO-CDR outperformed BMOPSO and Random algorithms in terms of Hypervolume and Maximum Spread (see tables IV and V). Thus, for Suite 2 the BMOPSO-CDR also covered a wider region of search space and obtained a larger Pareto front. However, considering the Spacing metric, all the algorithms were statistically equivalent. The BMOPSO was better than the Random algorithm only for the Hypervolume metric and they were statistically equal for Maximum Spread and Spacing. It is important to note that the Suite 2 is simpler than the Suite 1. Hence, it is easier to find good non-dominated solutions.

Tables VI and VII show the results considering the Coverage and Difference Coverage. Regarding the Coverage metric, it is possible to see that both BMOPSO and BMOPSO-CDR outperformed the Random approach, dominating approximately 97% and 94% (respectively for Suite 1 and Suite 2) of the non-dominated solutions found by Random. In contrast, the Random non-dominated solutions dominated approximately only 2% and 4% (for both suites) of non-dominated solutions found by the BMOPSO and BMOPSO-CDR algorithms. Different from the previous results, however, the BMOPSO was better than BMOPSO-CDR on both suites. In fact, this was the only metric in which BMOPSO outperformed the BMOPSO-CDR. The results of the Difference Coverage metric also show that the
BMOPSO-CDR approach outperformed BMOPSO and the Random method. It means that the BMOPSO-CDR covers larger regions of the search space that are not covered by the others.

V. CONCLUSION

In this work, we propose the use of multi-objective PSO for functional TC selection. The main contribution of the current work was to investigate PSO in a multi-objective way for selecting functional test cases considering both requirements coverage and execution effort. To best of our knowledge, multi-objective PSO was not yet investigated in the context of TC selection. We highlight that the developed method can be adapted to other test selection criteria and it is not limited to two objective functions. Furthermore, we expect that these good results can also be obtained on other application domains.

We implemented a Binary Multi-Objective PSO (BMOPSO) and a Binary Multi-Objective PSO using Crowding Distance and Roulette Wheel (BMOPSO-CDR) by merging two previous versions of the PSO algorithm. In the performed experiments, the BMOPSO-CDR and BMOPSO statistically outperformed the Random search approach for the majority of the evaluation metrics considered, thus verifying the viability of using them to TC selection. Furthermore, for the majority of the metrics, the BMOPSO-CDR outperformed the BMOPSO, showing that the CDR mechanism indeed improved the basic BMOPSO.

As future work, we cite to perform the same experiments on a higher number of test suites and to verify whether the obtained results are equivalent to the present work and whether these results can be extrapolated to other test’s scenarios other than mobile devices. Furthermore, we plan to determine whether the adopted metrics are indeed suitable to measure the quality of test suites and the behavior of them.

Also, we will investigate the impact of changing the PSO’s parameters in its performance on the TC selection problem. Besides, we intend to implement other multi-objective approaches (like Genetic Algorithms proposed by [9]) for a more complete comparison between techniques and to determine the distinct advantages of using PSO. Finally, we will investigate strategies to combine search techniques, in order to provide new hybrid algorithms for multi-objective TC selection.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES


