Multi-Objective Test Case Selection: A study of the influence of the Catfish effect on PSO based strategies

Luciano S. de Souza[1,2], **Ricardo B. C. Prudêncio**[2], **Flavia de A. Barros**[2]

[1]Federal Institute of Education Science and Technology of the North of Minas Gerais
IFNMG – Pirapora (MG), Brazil

[2]Center of Informatics (CIn)
Federal University of Pernambuco (UFPE)
Recife (PE), Brazil

luciano.souza@ifnmg.edu.br, {rbcp,fab}@cin.ufpe.br

Abstract. *During the software testing process many test suites can be generated in order to evaluate and assure the quality of the products. In some cases, the execution of all suites can not fit the available resources (time, people, etc). Hence, automatic Test Case (TC) selection could be used to reduce the suites 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). In this light, we developed mechanisms for TC selection in context of structural and functional testing. The proposed algorithms consider two objectives simultaneously: maximize branch coverage (or functional requirements coverage) while minimizing execution cost (time). These mechanisms were implemented by deploying multi-objective techniques based on Particle Swarm Optimization (PSO). Additionally, we added the so-called catfish effect into the multi-objective selection algorithms in order to improve their results. The performed experiments revealed the feasibility of the proposed strategies.*

1. Introduction

During the software development process, the software testing activities has grown in importance due to the need of high quality products. Software testing activities aim to assure quality and reliability of the developed products. Nevertheless, this activity is sometimes neglected, since it is very expensive, reaching up to 40% of the final software development cost [Ramler and Wolfmaier 2006]. In this scenario, automation seems to be the key solution for improving the efficiency and effectiveness of the software testing process.

The related literature presents two main approaches for testing: White Box (structural) or Black Box (functional) testing. Structural testing investigates the behavior of the software directly accessing its code. Functional testing, in turn, investigates whether the software functionalities are responding/behaving as expected without using any knowledge about the code [Young and Pezze 2005]. In both approaches, the testing process relies on the (manual or automatic) generation and execution of a Test Suite (TS), that is, a set of Test Cases. A Test Case (TC), consists of “a set of inputs, execution conditions, and a pass/fail conditions” [Young and Pezze 2005]. The execution of the TS aims to provide a suitable coverage of an adopted test adequacy criterion (e.g., code coverage, functional requirements coverage) in order to satisfy the test goals. In this scenario, both automatic generated or manually created TS may be large. In general, they try to cover all possible test scenarios in order to accomplish a good coverage of the adequacy criterion. Although it is desirable to fully satisfy the test goals, the execution of large suites is a very expensive task, demanding a great deal of the available resources (time and people) [Harold et al. 1993].

It is possible identify that large test suites contain some redundancies (i.e., two or more TCs covering the same requirement/piece of code). Hence, it is possible to reduce the suite in order to fit the available resources. This task of reducing a test suite based on a given selection criterion is known as *Test Case selection* [Borba et al. 2010]. TC selection is not easy or trivial since there may be a large number of TC combinations to consider when searching for an adequate TC subset.

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. Thus, the related literature focus on deploying automatic tools for TC selection.

A promising approach for the TC selection problem is to treat it as a search based optimization problem [Harman 2011]. This waym the search techniques explore the space of possible solutions searching for a TC subset in order to optimize a given objective function, i.e., the given selection criterion (see [Ma et al. 2005], [Yoo and Harman 2007], [Barltrop et al. 2010], [de Souza et al. 2010], [de Souza et al. 2013]). In most of the testing environments, there is not only a single selection criterion to be considered. In fact, it is very common the need to take into account, during the selection process, more than one criterion. Following this light, we can cite the use of multi-objective evolutionary approaches [Yoo and Harman 2007, Yoo and Harman 2010b, Yoo et al. 2011b, Yoo et al. 2011a] and the use of multi-objective Particle Swarm Optimization (PSO) techniques [de Souza et al. 2011] (our previous work).

In [de Souza et al. 2011], we investigated the multi-objective TC selection considering both the functional requirements coverage (quality), and the execution effort (cost) of the selected subset of TCs as objectives of the selection process. There, we developed the Binary Multi-Objective PSO techniques for TC selection by combining: (1) the binary version of PSO proposed in [Kennedy and Eberhart 1997]; and (2) the multi-objective PSO algorithms MOPSO [Coello et al. 2004] and MOPSO-CDR [Santana et al. 2009]. Despite the good results of the implemented techniques (BMOPSO-CDR and BMOPSO) on a case study, further improvements could be performed.

The use of the so-called "catfish" effect, for instance, has shown to improve the performance of the single objective version of the binary PSO [Chuang et al. 2011]. Hence, in the current work, we investigate whether this catfish effect can improve the performance of our former algorithms BMOPSO-CDR and BMOPSO [de Souza et al. 2011]. For that, we created two new multi-objective strategies, the CatfishBMOPSO-CDR and CatfishBMOPSO, by adding the catfish effect. We point out that the use of catfish effect, to the best of our knowledge, was not yet investigated in the context of test case selection or in the context of multi-objective optimization.

Additionally, in order to measure the performance of the proposed strategies, we performed experiments using two testing scenarios (structural and functional testing). For structural testing, we used test suites from five programs from the Software-artifact Infrastructure Repository (SIR) [Do et al. 2005], and for the functional testing, we used two unrelated test suites from the context of mobile devices. Each implemented algorithm returns to the user a set of solutions (test suites) with different values of branch coverage (for structural testing) or functional requirements coverage (for functional testing) *versus* execution cost. The user (tester) can then choose the solution that best fits into the available resources. It is important to highlight that, although the focus of our research is the multi-objective TC selection problem, the proposed techniques can also be applied to binary multi-objective optimization in other contexts.

Finally, it is also important to highlight that, to the best of our knowledge, there is no other work, in literature, that applied PSO techniques into the multi-objective TC selection problem (besides our previous work [de Souza et al. 2011]). Furthermore, we point out that no other multi-objective TC selection work performed both structural and functional testing in the same work. Hence, this is an promising study area, and this work aims to explore it a little further by creating new multi-objective strategies for test case selection.

The next section briefly presents search based concepts for TC selection. Section 3 explains our approach of multi-objective PSO for TC selection. Sections 3.2, 3.3 and 3.4 detail the implemented selection algorithms. Section 4 details the experiments performed to evaluate the proposed algorithms and presents the obtained results. Finally, Section 5 presents some conclusions and future work.

2. 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 costs is by reducing the size of test suites, since TS execution is resource consuming, and this approach is known as Test Case selection. Given an input TS, TC selection aims to find a relevant TC subset 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 seldom returns a representative TC subset.

A very promising approach for TC selection is to treat this task as a search optimization problem [Yoo and Harman 2010a]. In this approach, search techniques explore the space of possible solutions (subsets of TC), 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 and 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 objective (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 and El-Fakih 1999], conventional Genetic Algorithms [Ma et al. 2005], [Mansour and El-Fakih 1999] and PSO [de Souza et al. 2010]. Regarding multi-objective techniques, we highlight [Yoo and Harman 2007, Yoo and Harman 2010b, Yoo et al. 2011b, Yoo et al. 2011a, de Souza et al. 2011].

In this work, we developed two new multi-objective algorithms (the CatfishBMOPSO-CDR and CatfishBMOPSO) by adding the catfish effect into the BMOPSO-CDR and BMOPSO (see [de Souza et al. 2011]). Furthermore, the proposed algorithms were applied to select structural and functional test suites. The objective functions to be optimized were the branch coverage (for structural testing) or functional requirements coverage (for functional testing), and the execution cost (in time) of the selected TCs (for both approaches), in such a way that we maximize the first function (coverage) and minimize the second one (cost). The proposed algorithms return a set of non-dominated solutions (a Pareto frontier) considering the aforementioned objectives. 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).

It is not the purpose of this work to enter into a discussion concerning which objectives are more important for the TC selection problem. Irrespective of arguments about their suitability, branch or functional requirements coverage are likely candidates for assessing quality of a TS, and execution time is one realistic measure of cost.

Furthermore, we can point out some previous work that applied PSO in Software Testing, particularly for test case generation [Windisch et al. 2007] and for regression testing prioritization [Kaur and Bhatt 2011]. We can observe that the literature lacks works applying multi-objective PSO into the TC selection problem. For functional testing selection, we can only can only point out our previous work (see [de Souza et al. 2011]), and for structural testing selection, to the best of our knowledge, there is no previous work. Moreover, the catfish effect was not yet applied into a multi-objective context or into TC selection problem. Hence, this is an promising study area and we explore it a little further in this work by deploying new strategies that take this into account.

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

3. Multi-Objective PSO to Test Case Selection

In this work, we propose some Particle Swarm Optimization (PSO) based algorithms 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 $f_1(x), f_2(x), ..., f_k(x)$ 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 frontier.

The PSO algorithm is a population-based search approach, inspired by the behavior of birds flocks [Kennedy and Eberhart 1995] and has shown to be a simple and efficient algorithm compared to other search techniques, including for instance the widespread Genetic Algorithms [Eberhart and Shi 1998]. 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 ($\hat{\mathbf{t}}$) found by the particle (the memory of the particle);
4. the best position ($\hat{\mathbf{g}}$) found by the particle's neighborhood (the social guide of the particle).

For a number of iterations, the particles fly through the search space, being influenced by their own experience $\hat{\mathbf{t}}$ and by the experience of their neighbors $\hat{\mathbf{g}}$. Particles change position and velocity continuously, aiming to reach better positions and to improve the considered objective functions.

3.1. 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 $\mathbf{t} = (t_1, \ldots, t_n)$, in which $t_j \in \{0, 1\}$ indicates the presence (1) or absence (0) of the test case T_j within the subset of selected TCs.

As said, two objective functions were adopted. The coverage (code branches or functional requirements) objective (to be maximized) represents the amount (in percentage) of code branches (or functional requirements) covered by a solution **t** in comparison to the amount of covered by T. Formally, let $C = \{C_1, \ldots, C_k\}$ be a given set of k branches/functional requirements covered by the original suite T. Let $F(T_j)$ be a function that returns the subset of branches/functional requirements in C covered by the individual test case T_j. Then, the coverage of a solution **t** is given as:

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

In eq. (1), $\bigcup_{t_j=1}\{F(T_j)\}$ is the union of branches/functional 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 cost (the amount of time required to execute the selected suite). Formally, each test case $T_j \in T$ has a *cost score* c_j. The total cost of a solution **t** is then defined as:

$$Cost(\mathbf{t}) = \sum_{t_j=1} c_j \tag{2}$$

Finally, the MOO algorithms will be used to find a good Pareto frontier regarding the objective functions $C\_Coverage$ and $Cost$.

3.2. The BMOPSO-CDR algorithm

The Binary Multi-objective Particle Swarm Optimization with Crowding Distance and Roulette Wheel (BMOPSO-CDR) was firstly presented in [de Souza et al. 2011]. It uses an External Archive (EA) to store the non-dominated solutions found by the particles during the search process. See [de Souza et al. 2011] for more details of BMOPSO-CDR algorithm.

The following summarizes the BMOPSO-CDR:

1. Randomly initialize the swarm, evaluate each particle according to the considered objective functions and then store in the EA the particles' positions that are non-dominated solutions;
2. WHILE stop criterion is not verified DO
   (a) Compute the velocity **v** of each particle as:

   $$\mathbf{v} \leftarrow \omega\mathbf{v} + C_1 r_1(\hat{\mathbf{t}} - \mathbf{t}) + C_2 r_2(\hat{\mathbf{g}} - \mathbf{t}) \tag{3}$$

   where ω represents the inertia factor; r_1 and r_2 are random values in the interval [0,1]; C_1 and C_2 are constants. The social guide ($\hat{\mathbf{g}}$) is defined as one of the non-dominated solutions stored in the current EA.

   (b) Compute the new position **t** of each particle for each dimension t_j as:

   $$t_j = \begin{cases} 1, \text{ if } r_3 \leq sig(v_j) \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

   where r_3 is a random number sampled in the interval [0,1] and sig(v_j) is defined as:

   $$sig(v_j) = \frac{1}{1 + e^{-v_j}} \tag{5}$$

   (c) Use the mutation operator as proposed by [Coello et al. 2004];

   (d) Evaluate each particle of the swarm and update the solutions stored in the EA;

   (e) Update the particle's memory $\hat{\mathbf{t}}$;
3. END WHILE and return the current EA as the Pareto frontier.

3.3. The BMOPSO algorithm

The BMOPSO algorithm was created by merging the Multi-Objective Particle Swarm Optimization proposed by Coello Coello *et. al* [Coello et al. 2004] and the binary version of PSO proposed by Kennedy and Eberhart [Kennedy and Eberhart 1997].

Instead of using of using the crowding distance concept on EA's solutions, it generates hypercubes on EA's objective space and locate each solution within those hypercubes. Each hypercube has a fitness corresponding to the number of solutions located in it. Hence, when choosing the social guide $\hat{\mathbf{g}}$, a Roulette Wheel is used in order to favor the choose of a solution in a less crowded hypercube. For more details about the BMOPSO algorithm see [de Souza et al. 2011].

3.4. The CatfishBMOPSO-CDR and CatfishBMOPSO algorithms

The catfish effect derives its name from an effect that Norwegian fishermen observed when they introduced catfish into a holding tank for caught sardines. The introduction of a catfish, which is different from sardines, into the tank resulted in the stimulation of sardine movement, thus keeping the sardines alive and therefore fresh for a longer time [Chuang et al. 2011].

Similarly, the catfish effect is applied to the PSO algorithm in a such way that we introduce "catfish particles" in order to stimulate a renewed search by the rest of the swarm's particles (the sardines). Hence, these catfish particles helps to guide the particles, which can be trapped in a local optimum, to new regions of the search space, and thus leading to possible better areas [Chuang et al. 2011].

Chuang *et al.* [Chuang et al. 2011] proposed that, if the best value found by the swarm does not change after some iterations, it is considered stuck in a local optimum. Under such circumstances, 10% of particles with the worst fitness value are removed and substituted by catfish particles. The catfish particles are randomly positioned at extreme points of the search space, and stimulate a renewed search process for the particles of the swarm.

As the BMOPSO-CDR is an multi-objective algorithm, we needed to adapt the catfish effect in order to create the CatfishBMOPSO-CDR. For that, the following steps were introduced into the main loop (2) after step (e) of the BMOPSO-CDR algorithm:

```
1. IF the EA is not updated for 24 consecutive iterations[1] THEN
2. FOR the number of times equivalent of 10% of swarm size DO
     (a) Create a new solution $\mathbf{t}$
     (b) IF $r_4 < 0.5$ THEN
         $t_j = 1, \quad \forall t_j \in \mathbf{t}$
     (c) ELSE
         $t_j = 0, \quad \forall t_j \in \mathbf{t}$
         where $r_4$ is a random number sampled in the interval [0,1]
3. Randomly replace 10% of swarm's solutions by the catfish solutions.
```

In the single objective approach for the catfish effect (see [Chuang et al. 2011]), the authors needed to compare all the current solutions in order to find the worst solutions (particles) to be replaced by the new ones (catfish particles). Contrarily, as in our MOO approach we use an EA to store all best solutions (non-dominated solutions), we decided to keep the addition of the catfish effect as simple as possible by randomly choosing particles of current swarm to be replaced, instead of making comparisons to find the worst ones.

Additionally, the CatfishBMOPSO algorithm was created by adding the catfish effect into the BMOPSO algorithm using the same process of the CatfishBMOPSO-CDR.

[1]In [Chuang et al. 2011] they verify if the best value was not updated for 3 consecutive iterations, but, after preliminary trial and error parameter tuning, we discover that 24 is a better value for our context. Nevertheless, some future work is necessary in order to find the best value for our context.

4. Experiments and Results

This section presents the experiments performed in order to evaluate the search algorithms implemented in this work. The proposed multi-objective catfish approaches were compared whit their former algorithms in order to verify wether the addition of catfish effect has impact in the performance. For the structural testing scenario, the experiments were performed using 5 programs from SIR [Do et al. 2005]. In turn, the functional testing scenario, used test suites from the context of mobile devices[2].

4.1. Experiments Preparation

For the structural testing selection, initially we selected 5 programs from SIR: part of the *Siemens* suite (printtokens, printtokens2, schedule, schedule2), and the program space from the European Space Agency. These real world applications range from 374 to 6,199 lines of code. Since each program has a large number of available test suites we randomly selected 4 suites (referred here as T_1, T_2, T_3 and T_4) for each program. In Table 1 we give more information about the programs.

Table 1. Characteristics of the SIR programs

Program	Lines of Code	Avg. suite size
printtokens	726	16
printtokens2	570	17
schedule	412	10
schedule2	374	12
space	6199	157

The execution cost information was computed for each test case by using the Valgrind profiling tool [Nethercote and Seward 2003], as proposed in [Yoo and Harman 2007]. The execution time of test cases is hard to measure accurately due to the fact that it involves many external parameters that can affect the execution time as different hardware, application software and operating system. In order to circumvent these issues, we used Valgrind, which executes the program binary code in an emulated, virtual CPU. The computational cost of each test case was measured by counting the number of virtual instruction codes executed by the emulated environment and it allows to argue that these counts are directly proportional to the cost of the test case execution. Additionally, the branch coverage information was measured using the profiling tool **gcov** from the GNU compiler **gcc** (also proposed in [Yoo and Harman 2007]).

On the other hand, for the functional testing selection, we selected 2 test suites related to mobile devices: an Integration Suite (IS) and a Regression Suite (RS). Both suites have 80 TCs, each one representing an functional testing scenario. The IS is focused on testing whether the various features of the mobile device can work together, i.e., whether the integration of the features behaves as expected. The RS, 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. Contrarily to the structural suites, where each suite is intended to testing the related

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

program almost as whole, the used functional suites are related to a much more complex environment. Hence, just a little portion of the mobile device operational system is tested.

The cost to execute each test case of the functional suite was measured 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. Each TC has annotated which requirements it covers, thus we used this information in order to calculate the functional requirements coverage.

4.2. Metrics

In our experiments, we evaluated the results (i.e., the Pareto frontiers) obtained by the algorithms, for each test suite, according to four different quality metrics usually adopted in the literature of multi-objective optimization. The following metrics were adopted in this paper: Hypervolume (HV) [Deb and Kalyanmoy 2001], Generational Distance (GD) [Coello et al. 2007], Inverted Generational Distance (IGD) [Coello et al. 2007] and Coverage (C) [Deb and Kalyanmoy 2001]. Each metric considers a different aspect of the Pareto frontier.

1. Hypervolume (HV) [Deb and Kalyanmoy 2001]: 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.
2. Generational Distance (GD) [Coello et al. 2007]: The Generational Distance (GD) reports how far, on average, one pareto set (called PF_{known}) is from the true pareto set (called as PF_{true}).
3. Inverted Generational Distance (IGD) [Coello et al. 2007]: is the inverse of GD by measuring the distance from the PF_{true} to the PF_{known}. This metric is complementary to the GD and aims to reduce the problem when PF_{known} has very few points, but they all are clustered together. So this metric is affected by the distribution of the solutions of PF_{known} comparatively to PF_{true}.
4. Coverage (C) [Deb and Kalyanmoy 2001]: The Coverage metric indicates the amount of the solutions within the non-dominated set of the first algorithm which dominates the solutions within the non-dominated set of the second algorithm.

Both GD and IGD metrics requires that the PF_{true} be known. Unfortunately, for more complex problems (with bigger search spaces), as the *space* program, it is impossible to know PF_{true} a priori. In these cases, instead, a reference Pareto frontier (called here $PF_{reference}$) can be constructed and used to compare algorithms regarding the Pareto frontiers they produce [Yoo and Harman 2010b]. The reference frontier represents the union of all found Pareto frontiers, resulting in a set of non-dominated solutions found. Additionally, the C metric reported in this work refers to the coverage of the optimal set $PF_{reference}$, over each algorithm, indicating the amount of solutions of those algorithms that are dominated, e. g. that are not optimal.

4.3. Algorithms Settings

All the algorithms are run for a total of 200,000 objective function evaluations. The algorithms used 20 particles, ω linearly decreasing from 0.9 to 0.4, constants C_1 and C_2 of 1.49, maximum velocity of 4.0 and EA's size of 200 solutions. These values were the same used in [de Souza et al. 2011] and represent generally used values in the literature. As previously said, the catfish effect is applied whenever the EA not changes for 24 consecutive iterations.

4.4. Results

After 30 executions of each TC selection algorithm, the values of branch/functional requirements coverage and execution cost observed in the Pareto frontiers were normalized, since they are measured using different scales. All the evaluation metric were computed and statistically compared using the *Mann-Whitney U test*.

The *Mann-Whitney U test* is a nonparametric hypothesis test that does not require any assumption on the parametric distribution of the samples. It tests the null hypothesis that two independent groups are equal. In the context of this paper, the null hypothesis states that, regarding the observed metric, two different algorithms produce equivalent Pareto frontiers. The α level was set to 0.95, and significant *p-values* suggest that the null hypothesis should be rejected in favor of the alternative hypothesis, which states that the Paretor frontiers are different.

For the *Siemens* programs (printtokens, printtokens2, schedule and schedule2) we performed an exhaustive search and found the true optimal Pareto frontier. This frontier was compared with those produced by the selection algorithms, and we observed that, for all cases, the selection algorithms were capable to find the true Pareto frontier, thus the null hypothesis could not be rejected and, for the *Siemens* programs, all algorithms were statistically equivalent. This happened due to the fact of the search spaces of the *Siemens* programs were small, allowing the algorithms to easily find the optimal solution.

Additionally, the values of the metrics for the *Siemens* programs were not presented here because, as the values were equal, no important information can be retrieved from them. Contrarily, the space program has a bigger search space (an average of 2^{157} combinations) and the algorithms performed differently in some cases. Furthermore, for both space and mobile suites we do not performed exhaustive search due to the high computational cost.

Tables 2 and 3 present the average values of the adopted metrics (for the space and mobile suites), as well as the observed standard deviations. Additionally, asterisks (*) were used to indicate whenever two values are considered statistically equal (i.e. the null hypothesis was not rejected).

Regarding HV, IGD and C metrics, we can observe that the CatfishBMOPSO-CDR and CatfishBMOPSO outperformed the others. This means that: they dominated bigger objective space areas; were better distributed comparatively to optimal Pareto set (represented by the $PF_{reference}$); and were less dominated by the optimal Pareto set. However, regarding the GD metric, the catfish approaches worsened the former algorithms. It is possible to observe that, for the GD metric, the BMOPSO-CDR and BMOPSO outperformed the catfish approaches in almost all cases (except for the mobile regression suite where the CatfishBMOPSO was not outperformed). This means, that the former algorithms produce Pareto frontiers with better convergence to the optimal Pareto set.

Performing a deeper analysis on the produced Pareto frontiers, we observed that the frontiers produced by CatfishBMOPSO-CDR and CatfishBMOPSO had more solutions and were bigger than the frontiers produced by BMOPSO-CDR and BMOPSO. This happens because, when introducing the catfish particles into the extremities of the search space, it is expected that the catfish algorithms can find more solutions as well as find solutions near to the extremities. However, by introducing the catfish particles, the

Table 2. Mean value and standard deviation for the metrics - Program space

Metric	Suite	BMOPSO-CDR	BMOPSO	CatfishBMOPSO-CDR	CatfishBMOPSO
HV	T_1	0.804 (0.008)	0.799 (0.009)	0.864* (0.010)	0.861* (0.008)
	T_2	0.779* (0.007)	0.776* (0.007)	0.847** (0.006)	0.844** (0.010)
	T_3	0.815* (0.007)	0.812* (0.007)	0.871 (0.008)	0.864 (0.009)
	T_4	0.795 (0.007)	0.789 (0.008)	0.857 (0.008)	0.849 (0.008)
GD	T_1	0.005* (0.001)	0.004* (0.001)	0.012** (0.002)	0.013** (0.001)
	T_2	0.004* (8.8E-4)	0.004* (8.5E-4)	0.005 (8.8E-4)	0.004 (9.8E-4)
	T_3	0.004* (0.001)	0.004* (9.2E-4)	0.005 (9.2E-4)	0.004* (9.2E-4)
	T_4	0.003* (6.7E-4)	0.003* (5.9E-4)	0.003** (8.2E-4)	0.003** (8.0E-4)
IGD	T_1	0.030* (0.001)	0.030* (0.001)	0.004 (6.2E-4)	0.005 (0.002)
	T_2	0.027* (0.001)	0.027* (0.001)	0.003 (7.4E-4)	0.004 (7.4E-4)
	T_3	0.027* (0.001)	0.027* (0.001)	0.003 (5.7E-4)	0.004 (7.0E-4)
	T_4	0.026 (0.001)	0.028 (0.001)	0.003 (5.0E-4)	0.003 (6.8E-4)
C	T_1	0.99* (0.01)	0.97* (0.07)	0.94** (0.03)	0.95** (0.01)
	T_2	0.97* (0.04)	0.98* (0.03)	0.92** (0.02)	0.92** (0.01)
	T_3	0.97 (0.04)	0.98 (0.08)	0.91* (0.03)	0.91* (0.02)
	T_4	0.98* (0.02)	0.98* (0.02)	0.93** (0.02)	0.92** (0.02)

Table 3. Mean value and standard deviation for the metrics - Mobile suites

Metric	Suite	BMOPSO-CDR	BMOPSO	CatfishBMOPSO-CDR	CatfishBMOPSO
HV	IS	0.599 (0.005)	0.593 (0.005)	0.621* (0.003)	0.621* (0.004)
	RS	0.800 (0.008)	0.793 (0.007)	0.815* (0.005)	0.815* (0.006)
GD	IS	0.002* (3.2E-4)	0.002* (2.5E-4)	0.003 (2.4E-4)	0.002* (1.8E-4)
	RS	0.004* (6.1E-4)	0.004* (5.1E-4)	0.010** (0.002)	0.011** (0.001)
IGD	IS	0.007 (8.6E-4)	0.008 (8.9E-4)	0.002 (2.1E-4)	0.83 (0.01)
	RS	0.014 (0.002)	0.016 (0.002)	0.005* (0.001)	0.005* (0.001)
C	IS	0.98 (0.01)	0.97 (0.02)	0.96* (0.01)	0.95* (0.01)
	RS	0.98 (0.01)	0.97 (0.03)	0.96* (0.01)	0.95* (0.02)

algorithms seemed to lose some sort of capability to refine the already found solutions (the convergence was affected) and this directly impacted the GD metric.

Comparing the CatfishBMOPSO-CDR and CatfishBMOPSO, it is not possible to point out the best, since there is no clear domination between them. For most of the metrics they are statistically equal and sometimes one outperforms the other. In turn, the same can be said about the comparison between BMOPSO-CDR and BMOPSO. The results presented in [de Souza et al. 2011] differs from the presented here as there, the BMOPSO-CDR algorithm was never outperformed by the BMOPSO. This happened due to the use of different metrics to evaluate the algorithms. A future work is planned in order to address which metrics are the best for our context.

5. Conclusion

In this work, we propose the addition of the catfish effect into the binary multi-objective PSO approaches for both structural and functional TC selection. The main contribution of the current work was to investigate whether this effect can improve the multi-objective PSO approaches, from [de Souza et al. 2011], for selecting structural and functional test cases considering both branch/functional requirements coverage and execution cost. We highlight that the catfish effect in a multi-objective PSO was not yet investigated in the context of TC selection (or in any other context). We also point out that the developed selection algorithms can be adapted to other test selection criteria and are not limited to two objective functions. Furthermore, we expect that the good results can also be obtained on other application domains.

Expecting that the addition of catfish effect into multi-objective PSO could improve its performance, we implemented two catfish binary multi-objective PSO algorithms. We compared them with our former multi-objective PSO algorithms (the BMOPSO-CDR and BMOPSO). In the performed experiments, the CatfishBMOPSO-CDR and CatfishBMOPSO outperformed the others for all the metrics (except for the GD metric). Hence, we can conclude that the catfish effect indeed introduced improvements to the algorithms, but more research is necessary in order to prevent the negative impact to the algorithms' capacity of refine solutions. As future work we pretend to investigate the use of local search algorithms in order to deal with this drawback.

Another future work is to perform the same experiments on a higher number of programs in order to verify whether the obtained results are equivalent to those presented here and whether these results can be extrapolated to other testing scenarios other than the presented. Furthermore, we plan to determine whether the adopted metrics are indeed suitable to measure the quality of test suites.

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 add the catfish effect with other multi-objective approaches for a more complete comparison between techniques and to determine the distinct advantages of using the BMOPSO-CDR and BMOPSO as basis.

References

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

Barltrop, K., Clement, B., Horvath, G., and 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.*

Borba, P., Cavalcanti, A., Sampaio, A., and Woodcock, J., editors (2010). *Testing Techniques in Software Engineering, Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures*, volume 6153 of *Lecture Notes in Computer Science*. Springer.

Chuang, L.-Y., Tsai, S.-W., and Yang, C.-H. (2011). Improved binary particle swarm optimization using catfish effect for feature selection. *Expert Syst. Appl.*, 38(10):12699–12707.

Coello, C., Pulido, G., and Lechuga, M. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279.

Coello, C. A. C., Lamont, G. B., and van Veldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5. Springer.

de Souza, L. S., Miranda, P. B. C., Prudêncio, R. B. C., and Barros, F. d. A. (2011). A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In *In Proceedings of the 23rd International Conference on Tools with Artificial Intelligence (ICTAI 2011)*, Boca Raton, FL, USA.

de Souza, L. S., Prudêncio, R. B., de A. Barros, F., and da S. Aranha, E. H. (2013). Search based constrained test case selection using execution effort. *Expert Systems with Applications*, 40(12):4887 – 4896.

de Souza, L. S., Prudêncio, R. B. C., and Barros, F. d. A. (2010). A constrained particle swarm optimization approach for test case selection. In *In Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE 2010)*, Redwood City, CA, USA.

Deb, K. and Kalyanmoy, D. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 1 edition.

Do, H., Elbaum, S., and Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Softw. Engg.*, 10(4):405–435.

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

Harman, M. (2011). Making the case for morto: Multi objective regression test optimization. In *Fourth International IEEE Conference on Software Testing, Verification and Validation*, pages 111–114. IEEE Computer Society.

Harold, M. J., Gupta, R., and Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285.

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

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1942–1948.

Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, pages 4104–4109.

Lin, J.-W. and Huang, C.-Y. (2009). Analysis of test suite reduction with enhanced tie-breaking techniques. *Inf. Softw. Technol.*, 51(4):679–690.

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

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

Nethercote, N. and Seward, J. (2003). Valgrind: A program supervision framework. In *In Third Workshop on Runtime Verification (RVï¿½03).*

Ramler, R. and 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.*

Santana, R. A., Pontes, M. R., and Bastos-Filho, C. J. A. (2009). A multiple objective particle swarm optimization approach using crowding distance and roulette wheel. In *Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications*, ISDA '09, pages 237–242, Washington, DC, USA. IEEE Computer Society.

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

Yoo, S. and Harman, M. (2007). Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, pages 140–150.

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

Yoo, S. and Harman, M. (2010b). Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, 83:689–701.

Yoo, S., Harman, M., and Ur, S. (2011a). Highly scalable multi objective test suite minimisation using graphics cards. In *Proceedings of the Third international conference on Search based software engineering*, SSBSE'11, pages 219–236, Berlin, Heidelberg. Springer-Verlag.

Yoo, S., Nilsson, R., and Harman, M. (2011b). Faster fault finding at google using multi objective regression test optimisation. In *8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 11), Szeged, Hungary.*

Young, M. and Pezze, M. (2005). *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons.