

A MULTI-OBJECTIVE APPROACH FOR THE REGRESSION TEST CASE SELECTION PROBLEM

Camila Loiola Brito Maia

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Av. Paranjana, 1700, 60.740-903, Fortaleza, Ceará, Brazil
camila.maia@gmail.com

Rafael Augusto Ferreira do Carmo

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Av. Paranjana, 1700, 60.740-903, Fortaleza, Ceará, Brazil
carmorafael@gmail.com

Fabrcio Gomes de Freitas

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Av. Paranjana, 1700, 60.740-903, Fortaleza, Ceará, Brazil
fabriciogf.uece@gmail.com

Gustavo Augusto Lima de Campos

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Av. Paranjana, 1700, 60.740-903, Fortaleza, Ceará, Brazil
gustavo@larces.uece.br

Jerffeson Teixeira de Souza

Optimization in Software Engineering Group (GOES.UECE)
Natural and Intelligent Computing Lab (LACONI)
State University of Ceará (UECE)
Av. Paranjana, 1700, 60.740-903, Fortaleza, Ceará, Brazil
jeff@larces.uece.br

ABSTRACT

When software is modified, some functionality that had been working can be affected. The reliable way to guarantee that the software is working correctly after those changes is to test the whole system again, but generally there is not sufficient time. Then, it is necessary to select significant test cases to be executed, in order to guarantee that the system is working as it should be. Although there are already works regarding on the regression test case selection problem, some important features which can influence in the test case selection are not considered in them. In this work, we state a new and more complete multi-objective formulation for this problem. The work also shows the results of the solution for the problem using a multi-objective genetic algorithm, comparing it with a random algorithm.

KEYWORDS. Metaheuristics. Regression Test Case Selection. NSGA-II. MH.

1. Introduction

Software Test is a part of the entire software development process. The purpose of the testing process is to make sure computer code does what it was designed to do and that it does not do anything unintended (Myers, 2004). It is the ultimate review of specification, design, and code generation (Pressman, 2001).

The process of testing relies on the concept of *test case*, which is defined as a detailed specification of testing, including some important information about the test, like pre-conditions, post-conditions, data for entry, interdependency, etc. (Bastos, 2007). A group of test cases is named *test suite*.

During the lifetime of software it is common that changes on the software occur. Those changes may be necessary because of new requirements, adaptability issues or even correction of errors found after the release (*patches*). In such a situation, the test of the modified software system is required and, more important, crucial. This process of testing a system when just modifications have been done after the first final version is known as *regression testing*. The main reason to execute regression test is the inherent chance of introduction of errors in any human-based development activity.

There is only one reliable way to guarantee that the modified software is working as well as the previous version: testing the whole system again. Nevertheless, this approach is not practical if we consider the effort required to test all the system again when only a part of it had been modified. This scenario leads to a problem denominated *regression test case selection*. As its name suggests, this problem regards on the selection of significant test cases to be used in a later test process (i.e. a regression test process).

Many aspects can be considered for the regression test case selection problem, such as: available time to test, requirement importance to the client, test execution risks, requirement and code coverage, fault history, defect's severity, and so on. The most important issue is to combine some of these aspects to select the most significant test cases, i.e., the ones that better represent the client's desire and necessity. In this context, there are multiple objective functions to optimize as minimization of execution time of test cases, minimization of risk and maximization of importance of the requirements covered.

In the Operational Research field, metaheuristics are well known methods used to solve optimization problems including multiple objective (hereafter referred to as multiobjective) ones. So, it is enough convenient to use such methods to solve the regression test case selection. Since the regression test case selection is a difficult problem in the Software Engineering field, this problem-solution pair formulation is included in this relatively new field named Search Based Software Engineering (SBSE) (Harman et al., 2001). The SBSE field concerns on the solution of complex problems in the software development process which can be formulated as a search problem. To be able to solve such difficult problems, the SBSE field mainly relies on metaheuristic methods like Simulated Annealing, Tabu Search, Genetic Algorithm and others. The solution found by a metaheuristic procedure is not necessarily the best one, but it is a good solution for the problem. As it can be seen in Harman (2007), metaheuristics had been utilized to solve the test case selection problem, among others software engineering problems (Clarke et al., 2003).

This paper describes a multiobjective formulation for the regression test case selection problem, based on the authors's expertise on software testing, and analyses the results found by a multiobjective genetic algorithm (NSGA-II) on this problem.

The rest of this paper is organized as follows: Section 2 describes related works in the selection test case problem. Sections 3 and 4 detail the test case selection problem. Section 5 details the NSGA-II algorithm used in this paper. Finally, section 6 relates the experiment and section 7 shows the conclusion and proposes some future work.

2. Related Work

Chen et al. (2003) discuss a specification-based (black box) method for regression test selection. Code-based regression test selection is interesting, but as the size of the system grows it will be more difficult to do this test case selection by using code coverage. They present a function named RE (Risk of Exposure) which is based on both the probability of fault and the cost if a fault is executed. This RE function is applied to each test case and many scenarios. The scenarios that cover the most critical test cases and that have RE function with higher value are selected. In that work, the RE function is the only one objective function to be optimized in the regression test case problem.

Xu et al. (2005) propose a causal model to the test case selection problem. According to them, if the company implements the known test practice recommended by independent verification and validation process the unavailability of source code at system level is true. In addition, different test engineers may have different ideas to perform test case selection. The expert system had to integrate the different rules from the different test engineers. The following factors were considered as objectives: execution time, defect density and defect severity (based on past results). Higher priority is given to test cases that cover new and modified functions. The output is the system test plan, with the selected test cases and their execution order.

Yoo et al. (2007) introduce the concept of Pareto efficiency to the test case selection problem. The Pareto approaches take multiple objectives functions and construct a group of optimal solutions. In this case, each solution is the optimal test cases subsets. The paper instantiates the test case selection problem with two versions: the first one combines code coverage and cost as objective functions, and the second one combines code coverage, cost and fault history as objective functions. They implement three algorithms: a reformulation of Greedy Algorithm, the Non Dominating Sorting Genetic Algorithm (NSGA-II), and an island genetic algorithm variant of NSGA-II, which they call vNSGA-II. For small programs, the NSGA-II has the best performance, followed by the vNSGA-II algorithm. For larger programs, the greedy algorithm performs very well, outperforming the other algorithms. This was the first work about using multiple objective functions to the test case selection problem and using search based algorithms to solve them.

Mansour et al. (2001) compare five regression test selection algorithms, which include: Simulated Annealing, Reduction (Harrold et al., 1993), Slicing (Agrawal et al., 1993), Dataflow (Gupta et al., 1996), and Firewall (Mansour et al., 2001) Algorithms. The objective function of all of them is based on code coverage. The comparison is based on eight quantitative and qualitative criteria, for example, number of test cases, execution time and precision inclusiveness. They executed the algorithms for fifteen programs, from 21 to 381 lines of code.

3. Aspects Considered in the Problem

3.1. Importance of the Modified or Created Requirements

Each requirement has its own importance for the client. It is desirable that the most important requirements are tested first because they reflect the most important functionality or because they are the ones which have major risks for the client's business.

3.2. Dependence among Requirements

Some requirements have dependence with other requirements. Thereby, they have to be tested only if those requirements had already been tested. This specific information must be informed by the client.

3.3. Execution Time for Test Cases

Testing is usually the last activity of the software development process. However, the time available to the test phase is usually defined in the beginning of the development process. Besides, the test process (for being the last phase) may have to compensate the delays of previous phases.

There are two types of test cases: the manual test cases (executed manually by a human resource) and the automated ones (executed automatically). However, the time to execute each one of them does not have to be added. For example, suppose that there is one available human resource and that there are three test cases: two manual, and one automated. Assume that the first manual test case has execution time of 2 minutes, and that the second manual test case needs 1 minute to be finished. The third test case (the automated one) is executed in 2 minutes. The total time to the testing execution would be 3 minutes, because the automated test case does not need a human resource to execute it and it is executed in a parallel way. Otherwise, if the time to execute the automated test case was 5 minutes, the total time to the test execution would be 5 minutes, because this time is greater than the sum of the first two ones.

3.4. Risk of a Test Case

Each test case has risks related to itself. Some examples of risks of test cases are:

1. Risks of process, for example unavailable tools or documentation, not defined methodology, not qualified test analysts, etc;
2. Not available or only few resources;
3. Not available or not configured properly test environment;
4. Precarious configuration management;
5. No experience with new tools or technology used.

After the identification of the risks, it is important to identify both the risk probability and the impact of each risk to the client if that risk occurs. The table below, based on Bastos et al. (2007), demonstrates the relationship between the probability of the risk and its impact for client.

Table 1. Relationship between Probability of a Risk and Impact.

Impact for the Client's Business	Probability of the Risk		
	High	Medium	Low
High	HH	HM	HL
Medium	MH	MM	ML
Low	LH	LM	LL

To be able to use this information in a mathematical way, the values above were converted to numbers: HH = 9, HM = 8, HL = 7, MH = 6, MM = 5, ML = 4, LH = 3, LM = 2, and LL = 1.

By logical means, the greatest priority has to be given to the test cases which have the greatest impact for the client and also have the greatest probability of occur. Following this bias, Table 1 was "translated" to get a value that represents these characteristics of risk.

3.5. Available Time to Test

All software projects have a limited time to execute test over the application. This time is generally estimated in the beginning of the project, but it may change. The resources allocation have to be adapted to this limited time (in hours, for example) which is represented in this paper by T_{max} . This T_{max} can be transformed in d days (an approximation of the number of days), aiming to of calculate the available time to test for each resource.

3.6. Resources for the Test Case Execution

For each software project, there are human resources allocated to execute the test process. These resources may come from the project team itself or from an independent test team, for example. This resource is generally insufficient or limited. It also may happen that there is enough time to test all modified requirements, but there are not enough resource to do that work.

4. Formulation of the Problem

Let R be the set of all modified requirements and TC be the set of test cases that covers these requirements. TC must contain test cases that test the more significant requirements in less time. Based on the aspects described on the previous section, the problem is formulated as follows:

4.1. Requirements

Suppose that the number of requirements of the entire software application is q and that there are n requirements, modified or new, that have to be tested. This n amount vary from 1 to q , i.e., at least one requirement has to be tested (otherwise there is no sense in running tests) and the maximum amount of requirements to be tested is the total amount of requirements of the software. So, the set R of requirements that have to be tested can be described as:

$$R = \{r_i \mid i = 1, \dots, n\}, 1 \leq n \leq q$$

Each requirement has its importance, which varies from 1 (less importance) to 3 (more importance). The information about this importance value of the requirements must be informed by the client. The values can be represented by this function:

$$importance(r_i) = \{1, \dots, 3\}$$

Each requirement has also a set of preceding requirements, i.e. a group with other requirements that must be tested before this requirement. The function is:

$$precedings(r_i) = \{r \in R \mid r \text{ precedes } r_i\}$$

For each requirement there is a set of test cases associated, which contains the test cases that have to be executed to guarantee the requirement's functionality accorded with the client.

$$testCases(r_i) = \{t \in T \mid t \text{ covers } r_i\}$$

4.2. Test Cases and Test Suites

Suppose that the number of test cases of the entire software application is c and the set of all test cases is T . There are k test cases that have to be tested, based on the modified or new requirements. This number k vary from 1 to c . So, the set TC that contains all test cases that have to be executed is:

$$TC = \{t_j \mid j = 1, \dots, k\}, 1 \leq k \leq c$$

Each test case has a type (1 for manual or 2 for automated), an execution time, and a risk of execution. The value of this risk combines the probability which it can occur with the cost to the client if this risk occurs from 1 to 9 (as described in tables 1 and 2). The type and risk of a test case is represented by the following functions:

$$type(t_j) = \{1, 2\}$$

$$risk(t_j) = \{1, \dots, 9\}$$

We know the requirements covered by a test case using the function below:

$$requirements(t_j) = \{r \in R \mid r \text{ is covered by } t_j\}$$

Then, the following function is used to inform whether a determined test case covers a specified requirement:

$$cover(t_j, req) = \begin{cases} 1, & \text{if } req \in requirements(t_j) \\ 0, & \text{otherwise} \end{cases}$$

A solution for the regression test case selection problem is a test suite, i.e., a set of test cases, represented by TC , and the requirements covered by the test suite can be determined by:

$$RequirementsOfTestSuite(TC) = \{req \in R \mid \exists t_c \in T, cover(t_c, req)\}$$

The importance of a set of test cases is the sum of importance of all requirements that these test cases cover.

$$import(t_j) = \sum_{r_v \ni cover(t_j, r_v)} importance(r_v)$$

4.3. Resources

There are a limited number of people to execute the selected test cases, from 1 to m , here represented by:

$$P = \{p_h \mid h = 1, \dots, m\}$$

Each person has a working time and the extra overtime they are authorized to work. The working time and the extra overtime variables are used as input to calculate the available time to test, in hours. This information is combined to the execution time of test cases to limit the number of test cases selected to the executing phase.

It is important to consider that each resource has its productivity, represented here by letter d . For example, if the productivity is 80%, then $d = 0,8$.

$$availableTime(p_h) = workingTime(p_h) \times d + extraOvertime(p_h)$$

4.4. Mathematical Formulation of the Problem

Based on the subsections above, the mathematical formulation is:

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

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

3. Maximize $import(t_j)$

Subject to:

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

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

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

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

In this formulation, the functions indicated by the numbers 1, 2, and 3 are the objective functions, which correspond respectively to: minimization of the execution time, minimization of risk, and maximization of importance. The three lines indicated by the letters a, b, and c correspond to the constraints regarding on execution time, available time, and existence of some required test cases.

5. NSGA-II for Regression Test Case Selection

This section is intended to present an approach for the regression test case selection problem based on the NSGA-II metaheuristic.

5.1. Pareto-Optimal Solution

A solution $x^* \in \Omega$ is a Pareto-Optimal solution (an efficient solution) if there is no other solution $x \in \Omega$ such that $f(x) \leq f(x^*)$ and $f(x) \neq f(x^*)$. This efficient solution is not dominated by any feasible solution of the problem (Ferreira, 1999).

If $x^* \in \Omega$ is an efficient solution, then any alternative $x \in \Omega$ that provides a decrease in some objective, compared with x^* , has to provide an increase in another objective. Assuming that the solution of the problem must be efficient, the optimization process is reduced to the set of all efficient solutions of the problem (Ferreira, 1999).

5.2. The NSGA-II Metaheuristic

NSGA-II (Nondominated Sorting Genetic Algorithm II) is a multiobjective evolutionary algorithm proposed by Deb et al. (2002). The main feature of NSGA-II is that this metaheuristic alleviates some difficulties existent in other multiobjective algorithms, like nonelitism approach, computational complexity and the need for specifying a sharing parameter.

Initially, a random parent population P_0 of size N is created. Then it is sorted based on the nondomination. A fitness value (rank) is assigned to each solution with value equal to its nondomination level. A population Q_0 of size N is created based on P_0 , and using binary tournament selection, recombination and mutation. Thereafter, a combined population R_0 is formed with size $2N$. Then, the population R_0 is also sorted according to nondomination. The best solutions build the F_1 set, which is the first Pareto front. The F_2 set is built with the second best solutions, and so on. Finally, the nondomination sets are ordered as shown in Figure 1, and the N first elements are chosen for the new population.

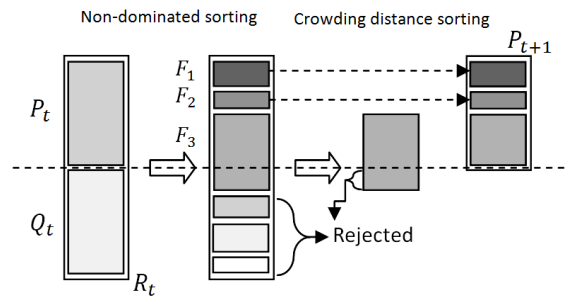


Figure 1. NSGA-II Procedure (Deb et al., 2002)

The diversity among nondominated solutions is introduced by using the crowding comparison procedure. The density estimation of solutions near a particular solution is calculated as follows: the average distance of two points on either side of this point along each objective was calculated. This quantity $I_{distance}$ is an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices. This is called crowding distance, and is illustrated in Figure 2.

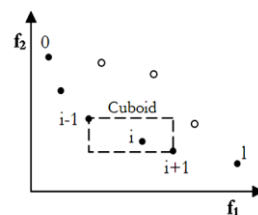


Figure 2. Crowding Distance (Deb et al., 2002)

For each objective function, the boundary solutions are assigned an infinite value. The overall crowding-distance value is calculated as the sum of individual distance values of each objective.

The new population P_{t+1} is used for selection, crossover and mutation to create a new population Q_{t+1} (Deb et al., 2002).

5.3. The NSGA-II Algorithm

The pseudo code below, in Figure 3, describes the NSGA-II algorithm.

```

 $R_t = P_t \cup Q_t$ 
 $F = \text{fast-non-dominated-sort}(R_t)$ 
 $P_{t+1} = \emptyset$  and  $i = 1$ 
until  $|P_{t+1}| + |F_i| \leq N$ 
     $\text{crowding-distance-assignment}(F_i)$ 
     $P_{t+1} = P_{t+1} \cup F_i$ 
     $i = i + 1$ 
Sort( $F_i, <_n$ )
 $P_{t+1} = P_{t+1} \cup F_i[1: (N - |P_{t+1}|)]$ 
 $Q_{t+1} = \text{make-new-pop}(P_{t+1})$ 

 $t = t + 1$ 

```

Figure 3. NSGA-II Pseudo code (Deb et al., 2002)

Initially, a set R_t is build by a parent and his offspring set. The Pareto front's sets are then formed, as explained in section 5.1.

The algorithm for the fast-non-dominated-sort function is detailed in figure 5. For each solution p , we calculate the number of solutions that dominates p (n_p) and a set of solutions that p dominates (S_p). Then, for each member q , if p dominates q then q is a member of S_p . Otherwise, n_p is added by one because p is dominated by q . If n_p is zero, p is not dominated by anyone, then p is included in F1 (first Pareto front). The others Pareto fronts are built in the same way.

So, while the new population P_{t+1} is not completely filled, elitism is applied and the elements of the nondominated sets (F1, F2, etc.) are added into P_{t+1} .

The crowding-distance-assignment algorithm, used for elitism, is in Figure 4. In this algorithm 'I' is a nondominated set. For each i in I and for each objective, we sort the population. The boundary solutions $I[1]$ and $I[L]$ are set to an infinite value. For all others solutions a value equal to the absolute normalized difference in the function values of two adjacent solutions is assigned (Deb et al., 2002).

```

crowding-distance-assignment(I)
 $l = |I|$ 
for each  $i$ , set  $I[i]_{\text{distance}} = 0$ 
for each objective  $m$ 
     $I = \text{sort}(I, m)$ 
     $I[1]_{\text{distance}} = I[l]_{\text{distance}} = \infty$ 
    for  $i = 2$  to  $(l - 1)$ 
         $I[i]_{\text{distance}} = I[i]_{\text{distance}} + (I[i + 1].m - I[i - 1].m) / (f_m^{\text{max}} - f_m^{\text{min}})$ 

```

Figure 4. crowding-distance-assignment Pseudo code (Deb et al., 2002)

```

fast-non-dominated-sort(P)
for each  $p \in P$ 
   $S_p = \emptyset$ 
   $n_p = 0$ 
  for each  $q \in P$ 
    if ( $p < q$ ) then
       $S_p = S_p \cup \{q\}$ 
    else if ( $q < p$ ) then
       $n_p = n_p + 1$ 
  if  $n_p = 0$  then
     $p_{rank} = 1$ 
     $F_i = F_i \cup \{p\}$ 
   $i = 1$ 
  while  $F_i \neq \emptyset$ 
     $Q = \emptyset$ 
    for each  $p \in F_i$ 
      for each  $q \in S_p$ 
         $n_q = n_q - 1$ 
        if  $n_q = 0$  then
           $q_{rank} = i + 1$ 
           $Q = Q \cup \{q\}$ 
     $i = i + 1$ 
     $F_i = Q$ 

```

Figure 5. fast-non-dominated-sort Pseudo code (Deb et al., 2002)

6. Empirical Evaluation

A series of empirical tests were executed, using the NSGA-II algorithm to generate the solutions for the regression test case selection problem. More specifically, the experiments were designed to answer the following question:

- I) Can a multiobjective function optimize the regression test case selection problem?
Does it generate an optimal solution?

6.1. Project Data Used

For the experiments, some data from a software project in a large company were collected.

The data was divided in:

- Requirement data: id (identification), description, importance, test cases which cover them, the requirements of which it is dependent, and an indicator of requirement added / changed
- Test cases: id, description, execution time (in minutes), risk, type (manual or automatic), and the requirements they cover.
- Resource: working time, productivity and extra overtime.
- Project: available time to test (in days).

There are 117 requirements for all system, but only 25 of them are new or changed requirements. There are 135 test cases to cover all system. Only one tester was allocated to execute the system test.

6.2. Experimental Design

Two algorithms were used for the experiments: the NSGA-II algorithm, configured in JMetal Framework (Durillo et al., 2006), and an implemented random algorithm.

For the NSGA-II algorithm, the following configurations were implemented:

- Random initial population.
- Number of iterations: 50 iterations were considered for the analysis, because we observed that from this value onward, the results do not differ. It was tested until get 250 generations.
- Crossover tax of 0.9, with Single Point Crossover method, and mutation tax of 1/199.
- Selection: Binary Tournament method.

The implemented random algorithm generates 50 valid solutions randomly.

In the experiment, the following three objectives were considered for the solutions (groups of test cases) found by the algorithms: risk (minimized), execution time (minimized), and importance (maximized). In addition, different sizes of population (number of solutions) were considered. For example, if the population is 20, then 20 solutions are generated by the algorithms. It was tested with the sizes 10, 20, 30, 40, and 50.

For technical reasons, we specify the environment of the experiment. All experiments were performed on the Linux Ubuntu 8.04 operating system. The hardware used was a Celeron M520 processor and 1 GB of memory.

6.3. Results

The results are presented in figures 6 and 7 below.

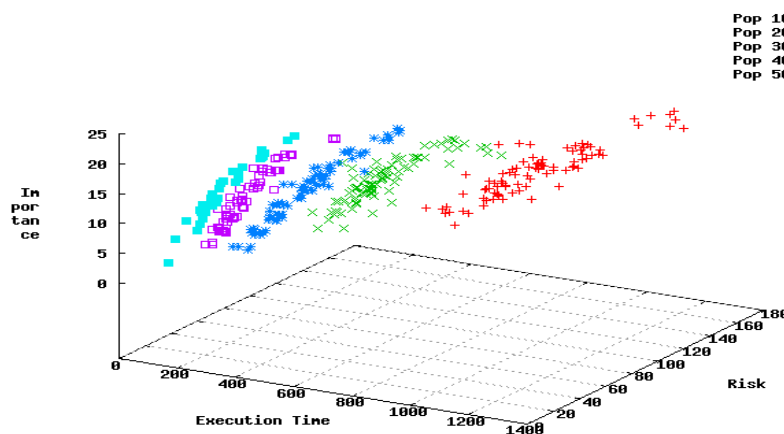


Figure 6. Risk x Execution Time x Importance

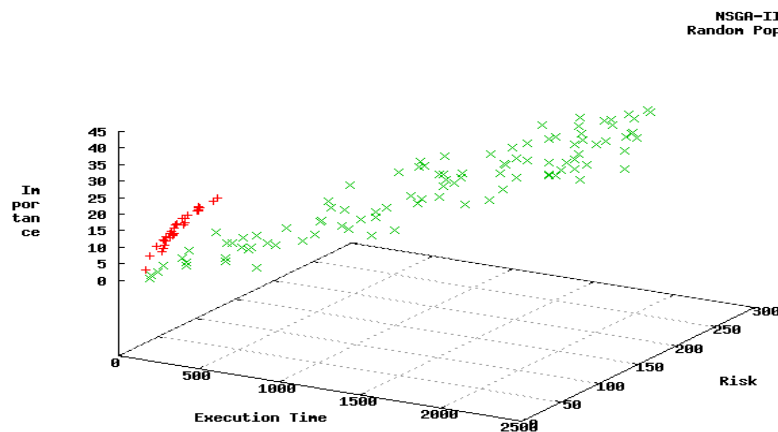


Figure 7. Comparing NSGA-II with a Random Algorithm

Figure 6 represents the generated populations for the NSGA-II algorithm with populations increasing 10 to 10 until 50 converging to a front. As already cited in this paper, from 50 onward, the results are the same. In figure 7, it is showed the comparison between NSGA-II algorithm (the red signs) and the random algorithm (the green signs) used to compare the results.

6.4. Analysis of Results

Analyzing the results obtained from the experiments, several relevant results can be pointed out. First, it can be observed that from a population of size 50 onward, the results are the same, i.e., the results converge to a front, generating the same solutions.

In order to validate the obtained results with NSGA-II algorithm, these results were compared with a random strategy. This comparison (see figure 7) shows that NSGA-II generates better solutions, considering the three cited objectives. The random algorithm does not consider any objective, which makes it generate poor solutions that do not improve the process of selection to the regression test process.

It can be observed in the graph that only two solutions randomly generated are better than the solutions generated by the NSGA-II algorithm. It is also important to mention that all solutions generated by the random algorithm are valid solutions for the regression test case problem. All the solutions generated by the NSGA-II algorithm were valid and optimal for the regression test case problem.

7. Conclusions and Future Work

The regression test case selection problem is an important component of the software development. The point of view of the client has always to be considered, as well as some development process aspects, like time to execute test. Each software engineer has an expertise about this problem, and we used our experience to model the problem as described in this paper.

This paper proposed a multiobjective formulation for the regression test case problem and executed it with the application of the NSGA-II algorithm (valid and optimal solutions found) and a random algorithm (valid solutions found), and we could understand that we can optimize the regression selection test case problem using a multiobjective formulation and algorithm. The NSGA-II showed to be a good solution for this problem, since the results found were optimal solutions.

As future work, a major quantity of data will be considered in the evaluation, and there will be some other multiobjective metaheuristics to validate the multiobjective formulation for the regression selection test case problem.

8. References

- Agrawal, H., Horgan, J. R., Krauser, E. W. and London, S. A.** (1993), Incremental Regression Testing, *Proceedings of Conference on Software Maintenance*, 348-357.
- Bastos, A., Rios, E., Cristalli, R. and Moreira, T.** (2007), *Base de Conhecimento em Teste de Software*, Martins Fontes Editora, 89-108.
- Chen, Y. and Probert, R. L.** (2003), A Risk-Based Regression Test Selection Strategy, *Chillarege Press*.
- Clarke, J., Dolado, J. J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M. and Shepperd, M.** (2003), Reformulating Software Engineering as a Search Problem, *Proceedings of IEE Software*, Vol. 150, Issue 3, 161-175.
- Deb, K., Pratap, A., Agarwal, S. and Meyarivan T.** (2002), A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, Vol. 6, Issue 2, 182-197.
- Durillo, J. J., Nebro, A. J., Luna, F., Dorronsoro, B. and Alba, E.,** JMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics, *Technical Report ITI-2006-10*, University of Málaga, (<http://neo.lcc.uma.es/staff/paco/pdfs/TECHREP%20ITI-2006-10.pdf>), 2006.
- Ferreira, P. A. V.** (1999), Otimização Multiobjetivo – Teoria e Aplicações, Tese de Livre Docência (<http://www.dt.fee.unicamp.br/~valente/teseLD.ps>).
- Gupta, R., Harrold, M. J. and Soffa, M. L.** (1996), Program Slicing-Based Regression Testing Techniques, *Software Testing, Verification and Reliability*, Vol 6, Number 2, 83-111.
- Harman, M.** (2007), The Current State and Future of Search Based Software Engineering, *Proceedings of the Future of Software Engineering (FOSE '07)*.
- Harman, M. and Jones, B. F.** (2001), Search-based software engineering, *Information and Software Technology*, 43, 833-839.
- Harrold, M. J., Gupta, R. and Soffa, M. L.** (1993), A Methodology for Controlling the Size of a Test Suite, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 2, Issue 3, 270-285.
- Mansour, N., Bahsoon R. and Baradhi G.** (2001), Empirical Comparison of Regression Test Selection Algorithms, *The Journal of Systems and Software*, 57, 79-90.
- Myers, G J.** (2004), *The Art of Software Test*,,
- Presman, R.S.** (2001), *Software Engineering*, McGraw Hill
- Xu, Z., Gao, K., and Khoshgoftaar** (2005) “Application of Fuzzy Expert System In Test Case Selection For System Regression Test, *Proceedings of Information Reuse and Integration (IRI '05)*.
- Yoo, S and Harman, M.** (2007), Pareto Efficient Multi-Objective Test Case Selection, *Proceedings of International Symposium on Software Testing and Analysis (ISSTA '07)*.