Generating Integration Test Orders for Aspect-Oriented Software with Multi-objective Algorithms

Wesley Klewerton Guez Assunção* [1]
Thelma Elita Colanzi[1] [2]
Silvia Regina Vergilio[1]
Aurora Trinidad Ramirez Pozo[1]

Abstract: O problema conhecido como CAITO refere-se à determinação de uma ordem para integrar e testar módulos que minimize os custos associados à construção de *stubs*. Algoritmos de busca, particularmente os algoritmos evolutivos, têm sido utilizados. Entretanto, o problema é bastante complexo e envolve diferentes fatores associados à construção de *stubs*, tais como diferentes medidas de complexidade, questões contratuais e outros. Estes fatores geralmente conflitam entre si e diferentes soluções para o problema são possíveis. Para lidar adequadamente com este problema, este trabalho explora o uso de algoritmos de otimização multiobjetivos. O artigo apresenta resultados da aplicação de dois algoritmos evolutivos - NSGA-II e SPEA2 - para o problema CAITO, em quatro sistemas reais implementados em AspectJ. Ambos algoritmos multiobjetivos são avaliados e comparados com o algoritmo de Tarjan e com um algoritmo genético mono-objetivo. Além disto, é apresentado um exemplo de como o testador pode utilizar as soluções obtidas de acordo com suas metas de teste.

Abstract: The problem known as CAITO refers to the determination of an order to integrate and test modules that minimizes stubbing costs. Search based algorithms have been used, mainly evolutionary ones. However, the problem is very complex since it involves different factors associated to the stubbing process, such as complexity measures, contractual issues and so on. These factors are usually in conflict and different possible solutions for the problem exist. To deal properly with this problem, this study explores the use of multi-objective optimization algorithms. The paper presents results from the application of two evolutionary algorithms - NSGA-II and SPEA2 - to the CAITO problem in four real systems, implemented in AspectJ. Both multi-objective algorithms are evaluated and compared with the traditional Tarjan's algorithm and with a mono-objective genetic algorithm. Moreover, it is shown how the tester can use the found solutions, according to the test goals.

*This is an extended version of the paper that appeared at LA-WASP 2011 (V Latin American Workshop on Aspect-Oriented Software Development and has been recommended to RITA.)

[1]Computer Science Department, Federal University of Paraná (UFPR), CP:19081, CEP: 19031-970, Paraná, Brazil
{wesleyk, silvia, aurora}@inf.ufpr.br
[2]Computer Science Department, State University of Maringá (UEM), Paraná, Brazil
thelma@din.uem.br

1 Introduction

Testing is a fundamental activity in the aspect-oriented (AO) software development related to quality assurance. This activity is generally conducted in different phases [1, 2, 3]. A first phase includes the test of each class, by testing its methods and crosscutting concerns. After this, an integration testing phase is performed to test interactions among modules that can be either classes or aspects. In many cases, a module A depends on a module B, i.e., to execute A it is required that B be available. When dependency cycles among modules exist, it is necessary to break the dependency and to construct a stub to emulate the behavior of module B.

The construction of stubs can be an expensive task and a recent study shows that is very common to find complex dependency cycles in Java programs [4]. In the AO context, new kind of relationships are possible between the modules, for example inter-type declarations that can impact in the stubbing costs. Many researches [5] found crosscutting concerns that are dependent on other crosscutting concerns, which implies a dependency between aspects, and between classes and aspects. To reduce integration test costs it is important to determine a sequence for integration and test of classes and aspects that minimizes the number of stubs to be constructed. The determination of such sequence is related to a problem called in the literature as CAITO (Class and Aspect Integration and Test Order [6]). This problem is not trivial, since different factors influence on the stub creation.

To solve the CAITO problem some strategies exist [5, 7, 6]. These strategies are based on solutions proposed for object-oriented (OO) programs. In the OO context a similar problem called CITO (Class Integration and Test Order) problem [8] has been intensively investigated and all knowledge acquired in such context has been extended for the AO context. Most of these extensions propose strategies based on graphs and classical algorithms [5, 7]. These strategies, however, do not present satisfactory solutions. Many times they produce local optimal since they do not analyze the consequences of breaking a dependency. Other disadvantage is that they need some extension to be used with other factors related to the stubbing process, for example: number of attributes of a class, number of calls or distinct methods invoked, constraints related to organizational or contractual reasons, etc.

To overcome these limitations, similarly to what happens in the OO context [9, 10], a strategy based on Genetic Algorithms (GA) has been applied [6]. GAs allow the use of different factors to establish the test orders by using a fitness function based on an aggregation of objectives to be minimized (a weighted average of number of operations and number of attributes). However, this fitness function requires the tester adjusts the weight of each objective, and the choice of the most adequate weights for the GA is a labor intensive task for complex cases.

To reduce these efforts and make the evolutionary strategy more practical for real sys-

tems, multi-objective optimization algorithms have been used to solve the problem in the OO context [11, 12, 13]. These studies present promising results when compared with a simple GA strategy. The algorithms allow the generation of more adequate solutions considering real constraints and diverse factors that may influence the CITO problem. In experiments conducted with different algorithms [12], the multi-objective evolutionary algorithms (MOEAs) presented the best results.

Motivated by the results obtained with MOEAs in the OO context, we explored in previous work [14, 15] a multi-objective search based approach for the CAITO problem. The approach treats the problem as a constrained combinatorial multi-objective optimization problem, more precisely, a problem where the goal is to find a test order set that satisfies constraints and optimizes different factors. NSGA-II [16] and SPEA2 [17] were used with a set of real AspectJ systems. The solutions obtained are evaluated according to Pareto concepts [18] and represent a good trade-off between different coupling measures. In this paper we resume the results obtained previously and present an evaluation of the multi-objective algorithms in comparison with two traditional ones: a graph-based algorithm (Tarjan's algorithm), and a mono-objective GA (a simple GA that uses an aggregated fitness function). The results show that the multi-objective algorithms present the best solutions, which are non-dominated, and its use is recommended mainly for complex cases.

The paper is organized as follows. Section 2 presents studies related to the CITO and CAITO problems. Section 3 reviews main concepts about multi-objective optimization and MOEAs. Section 4 introduces our multi-objective approach and Section 5 describes how the experimental evaluation was performed. Section 6 presents the results and a comparison of the algorithms. Section 7 contains some usage examples. Section 8 concludes the paper and discusses new perspectives for this study.

2 Related work

In the OO context, we can find in the literature several studies addressing the CITO problem [8, 19, 20, 21, 22, 23]. They are usually based on traditional algorithms and on directed graphs named Object Relation Diagrams (ORDs) [20, 23]. In such graphs the vertexes represent the classes, and their relations are represented by the edges. To determine an integration and test order, most strategies use the Tarjan's algorithm [24]. The algorithm is recursively applied in the graph to identify the cycles. The weight of each edge in the cycle is computed based on the number of incoming and outgoing dependencies. The cycle is broken by removing the edge with highest weight. When no more cycles remain in the graph, a reverse topological order is performed to determine the test order.

According to Briand et al [9] these strategies are very hard to be adapted to consider many factors that are involved in the stub creation, such as number of calls or distinct meth-

ods, constraints related to organizational or contractual reasons, etc. Other limitation of these strategies is that they focus on the reduction of broken cycles. However, there are cases where breaking two dependencies has a lower cost than breaking only one, and the found solutions can be sub-optimal.

To overcome this limitation, in another work, Briand et al [9, 10] explore the use of a GA and use fitness functions based on two coupling measures besides the dependency factor: the number of methods and attributes necessary for the stubbing process. A problem with the GA strategy based on the aggregation function is the choice of the weights for the objectives (coupling measures) being considered. This can be a labor-intensive task for complex cases.

The multi-objective approach was introduced in the OO context by Cabral et al [11]. They use Pareto concepts [18] to deal with the CITO problem, which is in fact multi-objective. In [11] the multi-objective approach based on the Ant Colony algorithm presents better solutions than the aggregated function used by the GA based approach of Briand et al. In addition to this, the multi-objective approach does not need weights adjustments and generate a set of good solutions that can be used by the tester. In [12] the authors evaluate the multi-objective approach with three different algorithms, and NSGA-II, the evolutionary one, obtained the best results.

As it happens with other problems found in the AOSD (Aspect-Oriented Software Development), the CITO problem was inherited from the OO context, and received the name CAITO (Class and Aspect Integration and Test Order). This allows the adaptation of the existing strategies for AO software. The existing strategies in the AO context are generally based on graphs too.

In the strategy of Ceccato et al [25] the classes are first tested without aspects. After this, the aspects are integrated and tested with the classes, and, at the end, the classes are tested in the presence of the aspects. The work of Ré et al [7] propose an extended ORD to consider dependency relations between classes and aspects. An example of extended ORD is presented in Figure 1, extracted from [5]. In the extended ORD the vertexes represent both classes and aspects. To a better visualization, the classes are on the left side of the figure and the aspects are on the right. Considering the aspects mechanisms, new relationships between vertexes are possible as shown below:

- Crosscutting Association (C) represents the association generated by a pointcut with a class method or other advice. In Figure 1 it is illustrated between the aspect `Billing` and class `Call`;

- Dependency (U) is generated by a relation between advices and pointcuts, and between pointcuts;

- Association Dependency (As) occurs between objects involved in pointcuts. This is shown in Figure 1 by the relationship between `Timing` and `Customer`;
- Inter-type Declaration Dependency (It) occurs when there are inter-type relationships between aspects and the base class. For example an aspect Aa declares that class A extends B. In the example there is this kind of dependency between `Billing` and `Local`; and among `MyPersistentEntities`, `PersistentRoot` and `Connection`;
- Inheritance Dependency (I) represents inheritance relationships between aspects or among classes and aspects, as it is observed by the aspects `PersistentEntities` and `MyPersistentEntities`, in Figure 1.

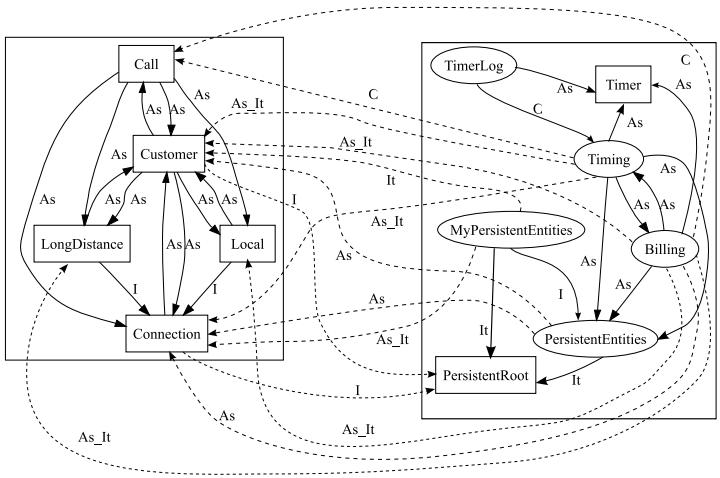


Figure 1. Extended ORD, extracted from [5]

The proposed ORD extension was evaluated in another work [5], in which four strategies were studied. They are: i) Combined: same strategy proposed by Briand et al [19] based on the Tarjan's algorithm, however applied to the extended ORD, combining the integration of aspects and classes; ii) Incremental+: applies the strategy of Briand et al [19], first integrating only classes and after following the order proposed by it, considering only aspects; iii) Reverse: applies the reverse combined order; and iv) Random: applies a random selected order. As a result of the study, the Combined strategy performed better than the Incremental+ strategy, producing a lower number of stubs. In fact, the Combined strategy seems to be more practical, since classes and aspects probably are tested together if both are under development.

The use of evolutionary algorithms in the AO context is recent. The work of Galvan et al [6] describes a simple mono-objective genetic algorithm that uses an aggregation of functions for the problem. This algorithm presents better solutions, considering attribute and operation complexities, than the strategies based on graphs proposed by Ré et al. However the simple GA strategy presents the same limitations that in the OO context: it requires adjustments for the objectives weights.

In spite of the promising results obtained by the MOEAs to solve the CITO problem, related work does not explore these algorithms in the AO context. Due to this, in Section 4, we describe an approach based on multi-objective optimization and Pareto concepts to solve the CAITO problem. The approach, first presented in [14], is based on the extended ORD proposed in [7] and can be used with all the strategies studied by Ré et al. However, to evaluate the approach (Section 5), we use the Combined strategy. To implement the approach we use MOEAs. These choices have their fundamentals on the aforementioned results presented respectively in [5] and [12].

3 Multi-objective optimization

Optimization problems with two or more objective functions are called multi-objective. In such problems, the objectives to be optimized are usually in conflict, which means that they do not have a single solution. In this way, the goal is to find a good "trade-off" of solutions representing a possible compromise among the objectives. The general multi-objective minimization problem with no restrictions can be stated as to minimize Equation 1.

$$\overrightarrow{f}(\overrightarrow{x}) = (f_1(\overrightarrow{x}), ..., f_B(\overrightarrow{x})) \tag{1}$$

subjected to $\overrightarrow{x} \in \Pi$, where: $\overrightarrow{x}$ is a vector of decision variables and Π is a finite set of feasible solutions.

Let $\overrightarrow{x} \in \Pi$ and $\overrightarrow{y} \in \Pi$ be two solutions. For a minimization problem, the solution $\overrightarrow{x}$ dominates $\overrightarrow{y}$ if Equations 2 and 3 are satisfied.

$$\forall f_i \in \overrightarrow{f}, i = 1...B, f_i(\overrightarrow{x}) \leq f_i(\overrightarrow{y}) \tag{2}$$

$$\exists f_i \in \overrightarrow{f}, f_i(\overrightarrow{x}) < f_i(\overrightarrow{y}) \tag{3}$$

$\overrightarrow{x}$ is a non-dominated solution if there is no solution $\overrightarrow{y}$ that dominates $\overrightarrow{x}$.

As mentioned, a multi-objective problem does not have a single solution. In the optimization process, a set of good solutions is found and forms the approximation to the Pareto

Front (PF_{approx}). Then, in this approximation we will find different non-dominated solutions.

Multi-objective optimization algorithms have been widely applied in several areas, such as Search Based Software Engineering [26], to solve problems with many interdependent interests (objectives) that must be optimized simultaneously. Variants of GA adapted to multi-objective problems were proposed. A GA is a heuristic inspired by the theory of natural selection and genetic evolution [27]. From an initial population, basic operators are applied consisting of selection, crossover and mutation. These operators evolve the population, generation by generation. Through the selection operator more copies of those individuals with the best values of the objective function are selected to be parent. So the best individuals (candidate solutions) will survive in the next population. The crossover operator combines parts of two parent solutions to create a new one. The mutation operator randomly modifies a solution. The descendent population created from the selection, crossover and mutation replaces the parent population.

Two representative MOEAs that are variants of traditional GAs are: NSGA-II (Non-dominated Sorting Genetic Algorithm) [28] and SPEA2 (Strength Pareto Evolutionary Algorithm) [17]. Each algorithm adopts different evolution and diversification strategies and were chosen in our study because both are well known and largely applied MOEAs [29]. Next, they are described briefly.

3.1 Non-dominated Sorting Genetic Algorithm (NSGA-II)

The algorithm NSGA-II [28] (see Figure 2) is a MOEA based on GA with a strong elitism strategy. For each generation, NSGA-II sorts the individuals from parent and offspring populations, considering the non-dominance, creating several fronts (lines 10 and 11 of Figure 2). The first front is composed by all non-dominated solutions. After removal of solutions belonging to the first front, the second front is composed with the solutions which become non-dominated. In the same way, the third front is formed by the solutions that become non-dominated after the removal of the solutions belonging to the first and the second fronts, and so on until all solutions are classified.

For the solutions of the same front, another sort is performed using the crowding distance to maintain the diversity of solutions (line 12 of Figure 2). The crowding distance calculates how far away the neighbors of a given solution are and, after calculation, the solutions are decreasingly sorted. The solutions in the boundary of the search space are benefited with high values of crowding distance, since the solutions are more diversified but with fewer neighbors.

Both sorting procedures, front and crowding distance, are used by the selection operator (line 17 of Figure 2). The binary tournament selects individuals of lower front. In case

```
Procedure NSGA-II
   Input: N', g, f_k(X) ▷ N' members evolved g generations to solve f_k(X)
 1 Initialize Population ℙ';
 2 Generate random population - size N';
 3 Evaluate Objectives Values;
 4 Assign Rank (level) based on Pareto - sort;
 5 Generate Child Population;
 6   Binary Tournament Selection;
 7   Recombination and Mutation;
 8 for i = 1 to g do
 9   for each Parent and Child in Population do
10     Assign Rank (level) based on Pareto - sort;
11     Generate sets of nondominated solutions;
12     Determine Crowding distance;
13     Loop (inside) by adding solutions to next generation starting from
       the first front until N' individuals;
14   end
15   Select points on the lower front with high crowding distance;
16   Create next generation;
17     Binary Tournament Selection;
18     Recombination and Mutation;
19 end
```

Figure 2. Pseudocode of NSGA-II (adapted from [29])

of same fronts, the solution with greater crowding distance is chosen. New populations are generated with recombination and mutation (line 18 of Figure 2).

The sorting procedures and the elitism strategy are illustrated in Figure 3, where P_t is the parent population; Q_t is the offspring population; F_1, F_2 e F_3 are fronts already sorted of combined populations (P_t and Q_t); and P_{t+1} is the population to be used in the next generation.

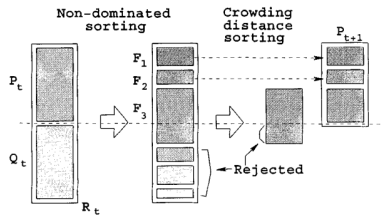


Figure 3. NSGA-II Flow diagram (extracted from [28])

3.2 Strength Pareto Evolutionary Algorithm (SPEA2)

The main distinction of SPEA2 [17] is the use of an external archive, in addition to its regular population, which stores non-dominated solutions found at each generation. The pseudocode of SPEA2 is presented in Figure 4. In each generation a strength value for all solutions is calculated and used by the selection operator. The strength value of a solution *i* corresponds to the number of *j* individuals, belonging to the archive and the population, dominated by *i*.

Procedure SPEA2

Input: $N', g, f_k(X) \triangleright N'$ members evolved g generations to solve $f_k(X)$

1 Initialize Population $\mathbb{P}'$;
2 Create empty external set $\mathbb{E}'$;
3 **for** $i = 1$ *to* g **do**
4 Compute fitness of each individual in $\mathbb{P}'$ and $\mathbb{E}'$;
5 Copy all nondominated individual from $\mathbb{P}'$ and $\mathbb{E}'$ to $\mathbb{E}'$;
6 Use the truncation operator to remove elements from $\mathbb{E}'$ when the capacity of the file has been exceeded;
7 If the capacity of $\mathbb{E}'$ has not been exceeded then use dominated individuals in $\mathbb{P}'$ to fill $\mathbb{E}'$;
8 Perform binary tournament selection;
9 Apply crossover and mutation;
10 **end**

Figure 4. Pseudocode of SPEA2 (adapted from [29])

The fitness of a solution is the sum of the strength values of all its dominators, from archive and population (line 4 of Figure 4). Value equal to 0 indicates a non-dominated individual, on the other hand, high values point out that the individual is dominated by many others. After the selection (line 8), new populations are generated with recombination and mutation (line 9).

In the evolutionary procedure, the external archive that is used in the next generation is filled with non-dominated solutions of the current archive and population (Figure 4, line 5). If the non-dominated front does not correspond exactly to the size of the archive, two cases are possible: in case of a too large new archive, a truncation procedure is performed (line 6 of Figure 4), in case of a too small new archive, the dominated individuals from the current archive and population are copied to the new archive (line 7 of Figure 4). For the truncation procedure, first, the distances from the solutions to their neighbors are calculated, after, the nearest neighbors are removed.

4 The multi-objective approach

In this section, we describe the multi-objective approach to determine an integration and test order for classes and aspects.

Regardless of the multi-objective algorithm used, first of all, it is necessary to obtain a representation (chromosome) for the problem. The choice of representation influences the implementation of all MOEAs stages. Since the CAITO problem is related to permutations of modules (classes and aspects), which form testing orders, the chromosome is represented by a vector of integers where each vector position corresponds to a module. The size of the chromosome is equal to the number of modules of each system. Thus, being each module represented by a number, an example of a valid solution for a problem with 5 modules is $(2, 4, 3, 1, 5)$. In this example, the first module to be tested and integrated would be the module represented by number 2. This representation is the same one used in our previous work [14].

To specify the kind of dependencies that should be considered, we adopted the ORD dependency model proposed by Ré et al [5, 7]. As related work, we consider that Inheritance and Inter-type declarations dependencies cannot be broken. This means that base modules must precede child modules in any test order t. The dependencies that cannot be broken are an input for the algorithms, called dependency matrix. In the algorithms, these constraints are checked during the generation of the initial population and in the application of the mutation and crossover operators. The treatment strategy involves a scan of the constraints from beginning to end of the chromosome, making sure that the precedence among the modules are not being broken. When a precedence constraint is broken, this module is placed at the end of the chromosome and all modules after the position are decremented by one;

The objectives to be minimized are related to a set of collected measures to be evaluated by the fitness function. As mentioned before, several possible measures and factors can be used in CAITO problem, such as coupling, cohesion and time constraints. We used the same coupling measures adopted by most related work [5, 6, 9, 11, 12]. They generally measure the dependencies between the server and client modules in terms of the number of attributes used and the number of distinct methods called.

So, considering that: (i) m_i and m_j are two coupled modules, (ii) modules are either classes or aspects, and (iii) the "operation" term represents class methods, aspect methods and aspect advices. We define:

a) **Attribute Coupling**: the number of attributes locally declared in m_j when references or pointers to instances of m_j appear in the argument list of some operations in m_i, as the type of their return value, in the list of attributes (data members) of m_i, or as local parameters of operations of m_i. This complexity measure counts the number of attributes

that would have to be handled in the stub if the dependency were broken. In the case of inheritance, we do not count the number of attributes inherited from the ancestor classes, as it occurs in the experiment of Briand et al [9]; and

b) **Operation Coupling**: the number of operations (including constructors) locally declared in m_j which are invoked by operations of m_i. This complexity measure counts the number of operations that would have to be emulated in the stub if the dependency were broken. In the case of inheritance, we count the number of operations declared in the ancestor modules.

The stubbing complexity of an order t is based on its attribute and operation coupling. Two complexities are then calculated in the following way:

- $A(t)$ (attribute complexity): The attribute complexity counts the maximum number of attributes that would have to be handled in the stub if the dependency were broken (attribute coupling measure). This information is an input for the algorithms and is represented by a matrix $AM(i, j)$, where rows and columns are modules and i depends on j. Then, for a given test order t and a set of d dependencies to be broken, the attribute complexity A is calculated according to Equation 4.

$$A(t) = \sum_{i=1,n} \sum_{j=1}^{n} AM(i, j); j \neq k \tag{4}$$

  Where n is the total number of modules and k is any module included before the module i, in test order t.

- $O(t)$ (operation complexity): The operation complexity counts the number of operations that would have to be emulated in the stub if the dependency were broken (operation coupling measure). This information is an input for the algorithms and is represented by a matrix $OM(i, j)$, where rows and columns are modules and i depends on j. Then, for a given test order t and a set of d dependencies to be broken, the operation complexity O is computed as defined by Equation 5.

$$O(t) = \sum_{i=1,n} \sum_{j=1}^{n} OM(i, j); j \neq k \tag{5}$$

  Where n is the total number of modules and k is any module included before the module i, in test order t.

Based on these constraints and measures presented above, the problem is the search for an order that minimizes two objectives: the attribute and operation complexities. In addition to this, it is necessary a strategy that states how the integration test is performed. In this work the strategy Combined is used.

5 Experimental evaluation description

This section describes how we conducted the empirical evaluation of the multi-objective approach. We adopted the same methodology of our previous work [14]. However, to get a better evaluation, we used two additional systems and performed a comparison with two traditional strategies based on the Tarjan's algorithm and on a simple GA. The multi-objective algorithms were re-executed. As a consequence, the results differ from the ones presented in [14], since the GA based algorithms are non deterministic. Below we present details about used systems, algorithm configurations and the quality indicators used in the evaluation.

5.1 Used systems

Four AO systems developed in AspectJ[3] were used in the study. AJHotDraw is an AO refactoring of the JHotDraw two-dimensional graphics framework. AJSHQLDB is also an AO refactoring of HSQLDB, which is a database manager developed in Java. HealthWatcher collects and manages public health related to complaints and notifications. TollSystem Demonstrator is a concept proof for automatic charging of toll on roads and streets.

The two objectives are related to the minimization of two coupling measures: number of methods and number of attributes. These measures are generally calculated during the software design, however it is difficult to obtain architectural design documentation of complex systems in order to execute empirical studies. So, reverse engineering was performed to identify the existing dependencies between modules from programs code. A parser based on AJATO[4] (AspectJ and Java Assessment Tool) was developed to do this. It uses the Java/AspectJ code as input and returns the syntactic tree code. From this tree, the associations, uses, inheritances, advices, pointcuts and inter-type declarations dependencies were identified. At the end, the parser generated as output three matrices (dependency and complexities) that were used as input to the MOEAs. Table 1 presents some information about these systems, such as number of classes, aspects and dependencies.

[3] AJHotDraw (version 0.4): http://sourceforge.net/projects/ajhotdraw/; AJHSQLDB (version 18): http://sourceforge.net/projects/ajhsqldb/files/; TollSystem (version 9): http://www.comp.lancs.ac.uk/ greenwop/tao/; HealthWatcher(version 9): http://www.aosd-europe.net/.

[4] http://www.teccomm.les.inf.puc-rio.br/emagno/ajato/

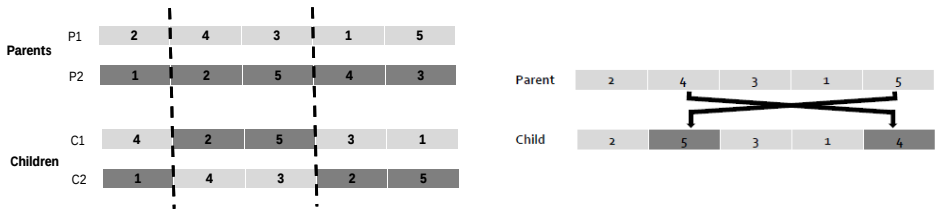
Table 1. Used Systems

System	Lines of Code	# Classes	# Aspects	# Dependencies						
				I	U	As	It	PointCuts	Advices	Total
AJHotDraw	18586	290	31	234	1177	140	40	0	1	1592
AJHSQLDB	68550	276	25	107	960	271	0	0	0	1338
HealthWatcher	5479	95	22	64	290	34	3	1	7	399
Toll System	2496	53	24	24	109	46	4	0	5	188

5.2 Algorithms

The Tarjan's algorithm used in the experiment is available at ANNAS Framework [30] and was implemented according to the strategy described in [5]. The GA used in the experiment is provided by [31]. It was adapted to compute the fitness based on the aggregation of both coupling measures and to work with the permutation problem. The approach was implemented with the evolutionary multi-objective algorithms NSGA-II and SPEA, available at the JMetal Framework [32].

We use the same problem representation and genetic operators to implement the evolutionary algorithms, the chromosome is represented by a vector whose positions assume an integer number that represents the modules. The size of this vector is equal to the number of modules of each system and a module must not appear twice in a test order. For the crossover operator we used the technique of Two Points Crossover. In this technique, two points are selected randomly, and the genes inside them are swapped in the children. The remaining genes are used to complete the solution, from left to right. Figure 5(a) shows an example of Two Points Crossover operator using an individual with 5 genes. For the mutation operator we used the technique of Swap Mutation. In this technique two genes are randomly selected and are swapped in the child. Figure 5(b) shows an example of Swap Mutation operator, using an individual with 5 genes.



(a) Two points crossover (b) Swap mutation

Figure 5. Evolutionary operators

The use of crossover and mutation operators can generate test orders that break the precedence constraints between the modules (dependencies of inheritance, aggregation and inter-type declaration). The strategy adopted to deal with these constraints consists of check-

ing the test order, and if an invalid solution is generated, the module in question is placed at the end of the test order.

The Tarjan's algorithm do not have parameters to be adjusted. The parameters of the MOEAs and GAs were adjusted empirically following the methodology adopted in our previous work [14].

To configure the mono-objective GA, besides the parameters related to the evolution process it was also necessary to set the weights of the measures: attribute and operation coupling to compose the aggregated fitness function. We have evaluated three combinations of weights. To verify the empirical influence of each measure on the stub construction we have used a configuration to minimize only the attribute coupling (identified here as the configuration GAA (GA with Attributes)). In this configuration the weight of the measure operation coupling was set to zero. The other configuration minimizes only the operation coupling (identified here as the configuration GAO (GA with Operations)). In this configuration the weight of the measure attribute coupling was set to zero. In the third configuration (configuration GA), equal importance was given to both measures.

Table 2 shows the parameters values adopted. Each evolutionary algorithm was executed 30 times for each AO system. The parameter Number of Fitness Evaluations was used as stopping criterion. All the algorithms executed the same number of fitness evaluations in order to analyze whether they can produce similar solutions when they are restricted to the same resources (number of fitness evaluations).

Table 2. MOEAs and GAs Parameters

Parameter	GAA	GAO	GA	NSGA-II	SPEA2
Population Size	300	300	300	300	300
Mutation Rate	0,2	0,2	0,2	0,02	0,02
Crossover Rate	0,9	0,9	0,9	0,95	0,95
Archive Size	-	-	-	-	250
Number of Fitness Evaluation	60000	60000	60000	60000	60000
Attribute Weight	1	0	0.5	-	-
Operation Weight	0	1	0.5	-	-

5.3 Quality indicators

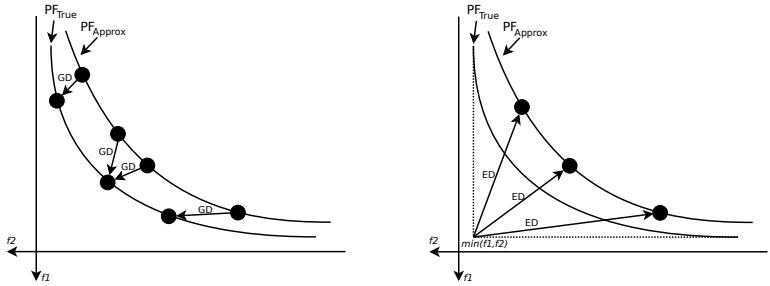
Two indicators were used to compare the multi-objective algorithms: Generational Distance (GD) and Euclidean Distance from the Ideal Solution (ED). These two indicators are used to measure the convergence to the Pareto Front (PF_{true}). At each run, an algorithm found an approximation set to the Pareto Front named PF_{approx}.

The Generational Distance (GD) indicator [33] calculates the average of the Euclidean

distance, in the objective space, between the solutions from an approximation of the Pareto Front (PF_{approx}) to the nearest solution of the Pareto Front (PF_{true}). So, GD is an error measure used to examine the convergence of an algorithm to the PF_{true}. For indicator GD, values closer to 0 are desired, since 0 indicates that all points of PF_{approx} are points on PF_{true}.

Euclidean Distance from the Ideal Solution (ED) is used to find the closest solution to the best objectives. An ideal solution has the minimum value of each objective, considering a minimization problem [34]. These minimum values are obtained from all non-dominated solutions. The purpose of this quality indicator is to find the closest solution to the ideal solution.

Calculating GD and ED requires the determination of PF_{true}. Considering that PF_{true} is not known, in our study, it was obtained by the union of the non-dominated solutions from all PF_{approx} found by all algorithms [35]. Figure 6(a) presents an example of GD and Figure 6(b) shows an illustrative example of calculating the ED indicator for a minimization problem with two objectives.



(a) GD (b) ED

Figure 6. Quality Indicators

The GD results were submitted to Wilcoxon test [36], by software R [37], in order to verify if the algorithms are considered statistically equivalent (*p-value* greater than the significance level α = 0.05).

6 Results and analysis

In this section the results of all algorithms are presented and analyzed. The comparison is accomplished considering the number of different and non-dominated solutions found by each algorithm.

The number of solutions found in all runs is shown in Table 3. The average of solutions found by each run is presented in parentheses. As mentioned before, the Tarjan's algorithm is deterministic, so it was run only once, and obtained just one solution for each system. All other algorithms were run 30 times. We can observe that the simple GA returned only one solution per run (average of 1.0). In this situation we observe the better performance of MOEAs, since they found a set of good solutions. For systems AJHotDraw and AJHSQLDB it is possible to verify that NSGA-II and SPEA2 found a set of solutions (with around to 4.0 and 30.0 solutions, respectively), helping the tester to make better decisions.

Table 3. Number of Solutions achieved by each algorithm

System	Tarjan	GAA	GAO	GA	NSGA-II	SPEA2
AJHotDraw	1 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	137 (4.6)	120 (4.0)
AJHSQLDB	1 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	991 (33.0)	777 (25.9)
HealthWatcher	1 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)
TollSystem	1 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)	30 (1.0)

To perform comparisons among the different algorithms, we obtained PF_{true}, composed by all solutions found in all runs of all algorithms, eliminating dominated and repeated ones. The costs of the solutions in the PF_{true} of each system are presented in Table 4. For example, the PF_{true} for AJHotDraw contains 11 solutions. The first order has an associated cost of 39 attributes and 66 operations, and so on.

Table 4. Cost of Solutions in the PF_{true}

System	Cost of Solutions (A, O)	Cardinality
AJHotDraw	(39,66),(42,23),(44,22),(45,21),(47,20),(50,18), (57,17),(75,16),(95,15),(117,14),(130,13)	11
AJHSQLDB	(1350,403),(1366,394),(1379,390),(1389,380),(1407,374),(1430,366), (1444,357),(1462,352),(1473,351),(1485,343),(1503,338),(1514,337), (1524,335),(1525,332),(1528,331),(1539,330),(1543,326),(1545,324), (1556,323),(1562,322),(1563,319),(1574,318),(1577,317),(1584,311), (1595,310),(1598,309),(1610,305)(1616,304)	28
HealthWatcher	(9,2)	1
TollSystem	(0,0)	1

We can notice that AJHSQLDB is the most complex system with 28 different possibilities to integrate and test its modules. AJHotdraw is the second one. HealthWatcher and TollSystem are the simplest, since only one integration and test order was found for each system, requiring few or none stub.

The cost of the solutions obtained by each algorithm (PF_{Approx}) is presented in Table 5. The solutions of PF_{true} are highlighted in boldface.

For AJHotDraw, the PF_{true} set is formed from one solution found by GAA, two

Table 5. Cost of solutions in PF_{Approx} achieved by each algorithm

System	Tarjan	GAA	GAO	GA	NSGA-II	SPEA2
AJHotDraw	(52,23)	**(39,66)**	**(117,14)**	(67,21)	(42,26)	**(42,23)**
		(47,52)	**(130,13)**	(79,16)	(43,25)	**(44,22)**
					(45,24)	**(45,21)**
					(46,23)	**(47,20)**
					(47,22)	**(50,18)**
					(48,21)	**(57,17)**
					(53,20)	**(75,16)**
					(55,19)	**(95,15)**
					(59,18)	
					(77,17)	
					(95,16)	
AJHSQLDB	(1690,346)	(1532,753)	(2365,359)	(1666,520)	**(1350,403)**	(1381,470)
		(1543,709)	(2645,337)	(1696,372)	**(1366,394)**	(1382,449)
		(1607,707)	(2735,333)		**(1379,390)**	(1395,410)
		(1616,682)	(2853,307)		**(1389,380)**	(1401,408)
					(1407,374)	(1402,407)
					(1430,366)	(1404,404)
					(1444,357)	(1407,402)
					(1462,352)	(1411,401)
					(1473,351)	(1414,399)
					(1485,343)	(1417,397)
					(1503,338)	(1424,396)
					(1514,337)	(1427,393)
					(1524,335)	(1430,380)
					(1525,332)	(1433,378)
					(1528,331)	(1434,375)
					(1539,330)	(1448,371)
					(1543,326)	(1450,370)
					(1545,324)	(1452,369)
					(1556,323)	(1468,364)
					(1562,322)	(1475,363)
					(1563,319)	(1476,361)
					(1574,318)	(1490,353)
					(1577,317)	(1507,349)
					(1584,311)	(1509,348)
					(1595,310)	(1518,347)
					(1598,309)	(1533,339)
					(1610,305)	(1550,335)
					(1616,304)	(1551,334)
						(1573,330)
						(1574,326)
						(1576,325)
						(1593,320)
						(1613,316)
						(1615,315)
						(1629,310)
HealthWatcher	(14,22)	(9,11)	(32,2)	**(9,2)**	**(9,2)**	**(9,2)**
TollSystem	**(0,0)**	(0,1)	(4,0)	**(0,0)**	**(0,0)**	**(0,0)**

solutions found by GAO and eight solutions found by SPEA2. For AJHSQLDB, NSGA-II found all the twenty eight solutions that compose the PF_{true}. For HealthWatcher, three algorithms found the solution that is the PF_{true}. They are: GA, NSGA-II and SPEA2. These same algorithms and also Tarjan's algorithm found the solution in PF_{true} for TollSystem.

It seems that for TollSystem and HealthWatcher the objectives in question are not highly interdependent and conflicting, since the GA with equal weights get the best results. Regarding to HealthWatcher, the value achieved for measure A is 9 and for measure O is 2. So, the solution found by MOEAs implies in making stubs for emulating dependencies with 9 attributes and 2 operations, which can be methods of classes/aspects or advices of aspects. It seems that the algorithms can achieve a single solution for this software due to its low number of cycles (1).

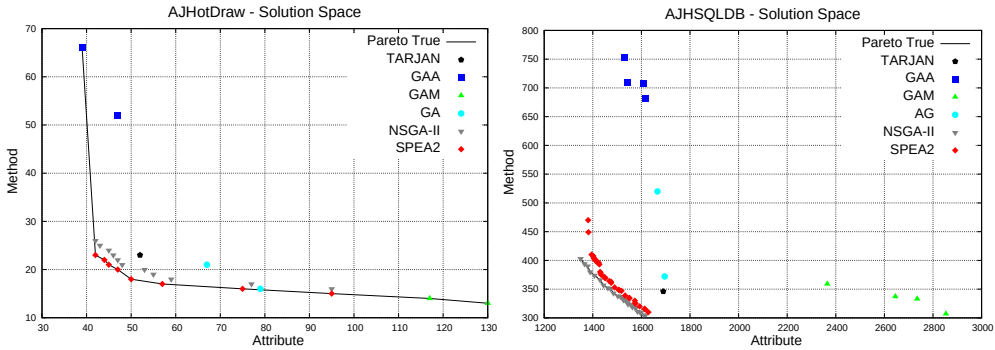
In the case of TollSystem, both MOEAs, GA and Tarjan's algorithm achieved the optimal solution, which consists of an order of modules that does not need any stub to perform the integration test. This system has only eight cycles, so to find the optimal solution is easy for the algorithms. However, considering that this system has 53 classes and 24 aspects, to find the best order to test these modules to minimize the number of stubs is not trivial for any tester. So, the use of an automatic approach is an interesting alternative to help in this work.

For more complex systems, apparently there is not a direct relationship between the number of modules/dependencies and the size of solutions set found by the algorithm, since AJHotDraw has more modules and more dependencies than AJHSQLDB but the algorithms found a lower number of solutions to the former.

Comparing the different algorithms (Tarjan, GAs and MOEAs) based on the results of Table 5, we can state that MOEAs are more suitable to solve this problem. The MOEAs found 70% of the solutions that compose the PF_{true} of system AJHotDraw, 100% of the solutions that compose the PF_{true} of system AJHSQLDB, and also found the unique solution of the systems TollSystem and HealthWatcher.

Figures 7(a) and 7(b) show the distribution of solutions of PF_{true} in the objective space for AJHotDraw and AJHSQLDB, respectively. Both graphics include the two objectives used in the experiment (measures A and O). By analyzing the solutions which form the PF_{true} and the distribution of solutions in the search space, we can verify which algorithms have better behavior. It can be observed the similarity of the solutions found by NSGA-II and SPEA2, since they are located in the same area of the objective space. In the case of solutions found by Tarjan's algorithm and GAs, the solutions are more spread in the objective space and not so close to PF_{true}. Tarjan's algorithm found only a single solution, then it does not offer different possibilities to the tester. Among the three GAs, the GA that has the same weight for the two measures seems to be the best, since its solutions present a better balance between the measures an they are closer to the minimum objectives, representing

some interesting possibilities to the tester. By observing the solutions in the objective space (Figures 7(a) and 7(b)), it is possible to note that the solutions found by GAA and GAO are located in the extremities. So, they do not have good balance between the measures.



(a) AJHotDraw (b) AJHSQLDB

Figure 7. Solution Space

The MOEAs found the largest number of solutions in the PF_{true} for the complex systems (AJHotDraw and AJHSQLDB), and they have found the best solutions in the search space. These solutions are better because they are in the region that offers the best trade-off to the tester.

Then, we can state that, despite using different evolutionary strategies, NSGA-II and SPEA2 are more effective to solve the CAITO problem than the traditional algorithms when these two objectives are used. The returned set contains a variety of good solutions that represent a good trade-off between the used coupling measures, even for complex cases. Since the MOEAs are better to solve CAITO problem with two objectives, in the next section both MOEAs are compared using the quality indicators GD and ED.

6.1 Comparing the MOEAs

In this section the values obtained for the two quality indicators are presented and analyzed. Table 6 presents the GD indicator values. The values are very similar, but NSGA-II reached better average for AJHotDraw and AJHSQLDB. For the systems with only one solution (HealthWatcher and TollSystem), the values of GD are 0 because in all runs the two MOEAs achieved the same solution.

However, Wilcoxon test returned a $p - value = 0.6543$ for AJHotDraw and a $p - value = 0.002468$ for AJHSQLDB for indicator GD. Thus, the values indicate significant

Table 6. Average and Standard Deviation for GD

System	NSGA-II		SPEA2	
	Average	Standard Deviation	Average	Standard Deviation
AJHotDraw	1.287860	0.758893	1.353876	0.890973
AJHSQLDB	0.129224	0.082192	0.241882	0.171367

difference only for AJHSQLDB. These differences between NSGA-II and SPEA2 are presented in Figure 8, where we can observe that NSGA-II has better average than SPEA2.

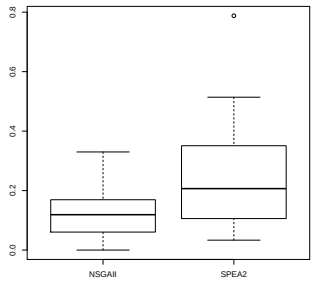


Figure 8. Boxplot of GD for the AJHSQLDB system

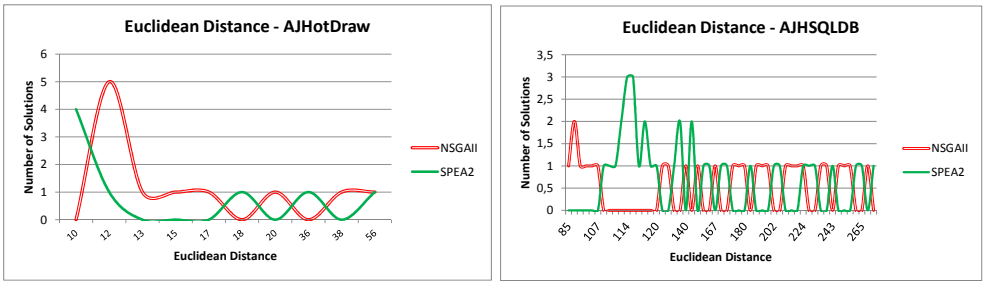
To calculate ED it is necessary to determine the cost of the ideal solution achieved by the algorithms for each system. These costs of the ideal solutions are presented in the second column of Table 7. This table also presents the lowest ED and the fitness of the associated solution, in the fourth and fifth columns respectively. The MOEA that achieved the lowest ED is typed in boldface. We can note that SPEA2 achieved the solutions that have the lowest ED from the ideal solution for AJHotDraw. For AJHSQLDB the solution with lowest ED was achieved by NSGA-II. In the case of HealthWatcher and TollSystem both MOEAs achieved the ideal solution since they found a single solution.

Figures 9(a) and 9(b) show ED from ideal solution of each solution found by MOEAs. For AJHotDraw, SPEA2 found a greater number of solutions with lower ED. For AJHSQLDB, NSGA-II found a greater number of solutions with lower ED.

Despite NSGA-II and SPEA2 use different ways to explore the solution space, they achieve similar feasible solutions. The returned set contains a variety of good solutions that represent a good trade-off between the used coupling measures. From these results, it is possible to affirm that NSGA-II and SPEA2 are efficient for CAITO problem with two objectives, since none had better results than the other in all systems.

Table 7. Costs of the Ideal Solution and the lowest distances found

Software	Cost of the Ideal Solution	MOEA	Lowest Achieved ED	Fitness of the Solution
AJHotDraw	(39,13)	NSGA-II	12.0416	(47,22)
		SPEA2	**10.0000**	**(45,21)**
AJHSQLDB	(1350,304)	**NSGA-II**	**85.4225**	**(1389,380)**
		SPEA2	109.9864	(1434,375)
HealthWatcher	(9,2)	**NSGA-II**	**0**	**(9,2)**
		SPEA2	**0**	**(9,2)**
TollSystem	(0,0)	**NSGA-II**	**0**	**(0,0)**
		SPEA2	**0**	**(0,0)**



(a) AJHotDraw (b) AJHSQLDB

Figure 9. Number of Solutions X Euclidean Distance

However, NSGA-II has a slight better convergence than SPEA2 as presented by GD indicator. NSGA-II also has good solutions near to the ideal solution, as the ED indicator shows. Often, decision makers prefer solutions near to the ideal solution. So, in this case the NSGA-II should be chosen.

7 Usage examples

The use of the multi-objective approach to solve the CAITO problem is feasible and efficient as discussed above. It allows the automatic determination of good test orders. However, the configuration of the algorithms can be difficult for the tester, who usually do not know about multi-objective algorithms. To overcome this limitation, references values found in the literature and that ones from Section 5.2 can be used. Other limitation is related to the use of the approach, related to the selection of an order to be used during the test. In this section we illustrate possible examples of how to select such order. The tester can choose among

them according to some objective that he/she needs to prioritize and other factors associated to the development process.

In order to illustrate this, consider some NSGA-II and SPEA2 solutions for AJHSQLDB, presented in Table 8. The solution a has the lowest ED and the solution b is an alternative since it has the second lowest ED. But, how may the tester choose among them? If the solution a of NSGA-II were chosen for testing AJHSQLDB (Table 8), it would be necessary to create stubs for emulating dependencies of 1389 attributes and 380 operations. On the other hand, if the solution b of the same algorithm were chosen it would be necessary to create stubs for emulating dependencies of 1407 attributes and 374 operations. Remember that these measures were defined in a generic way to be used together with AO software. Thus, the operation term is related to class methods, aspect methods or aspect advices.

Table 8. Some Solutions achieved for AJHSQLDB

	NSGA-II			SPEA2			Minimum Objectives
	A	**O**	**ED**	**A**	**O**	**ED**	
a	1389	380	85.42	1434	375	109.98	(1350, 304)
b	1407	374	90.27	1430	380	110.34	

So, in the first case, the prioritized objective would be the measure A and, in the second case, the measure O. Solutions a and b have the best trade-off between these objectives and, as a consequence, they have lower ED than the solutions that contain the lowest value to measures A and O.

Since both algorithms achieved a single solution for HealthWatcher and TollSystem, anyone can be used and the tester does not need to establish criteria for choosing the order of integration test because the best sequence is known (solution generated by the algorithms).

Although in this study the analysis has considered two MOEAs, in real life, the tester probably would have results of only one MOEA. From the obtained solutions, the tester must decide which objective(s) to prioritize and sort the solutions according to the priorities. The decision on the prioritization of objectives can be influenced by several factors, such as restrictions on business, economics, etc.

In our study, the Combined strategy was used. Hence, all solutions returned by the algorithms contain orders to integration test of classes and aspects together. However, another strategy could be used, for instance, the integration of only classes and, after this, aspects. Then the tester could use the solutions in a different way.

8 Conclusions

This paper presented an approach in which MOEAs are applied to solve the CAITO problem. To evaluate such approach, an experimental study was performed with NSGA-II and SPEA2 algorithms, and four AO systems: AJHotDraw, AJHSQLDB, HealthWatcher and TollSystem. In this study the MOEAs aimed at minimizing two objectives related to attribute and operation complexities. Furthermore, the multi-objective algorithms were compared with the Tarjan's algorithm and with a simple and mono-objective GA with three different fitness functions.

Although the CAITO problem is multi-objective for general cases, HealthWatcher and TollSystem are examples of particular cases in which the objectives are not in conflict, and for this reason all the algorithms have achieved a single solution. For both systems, the simple GA with equal weights for the objectives reaches better solutions than GAO and GAA.

Moreover, the other two systems (AJHotDraw and AJHSQLDB) consist of typical examples in which the measures are in conflict. Indeed they are more complex than the others. They have a lot of dependencies and consequently many cycles. For such systems the multi-objective approach achieved the best results. The multi-objective algorithms find a set of different solutions containing different alternatives of compromise between the objectives. For AJHSQLDB both traditional algorithms do not obtain solutions in the PF_{true}, which are composed by non dominated solutions. From the analysis of multi-objective algorithms according to the quality indicators, it is possible to affirm that NSGA-II has a slight better convergence than SPEA2, as the GD indicator suggests. The NSGA-II also has good solutions near the ideal solution, as the ED indicator shows.

As future work, different configurations of the approach can be explored. For example other cost models can be used, representing other objectives for the optimization algorithm, as well as, other strategies for integrating classes and aspects, such as the incremental one. In addition to this, experiments with other AO systems to confirm the evidences found in this work should be conducted. Systems with a greater number of modules should be used to better evaluate scalability. Other possible extension is the implementation of the approach with other MOEAs, such as PAES and MOCell. Currently, we are extending the approach to make it more generic and applicable to other contexts such as software development oriented to components, product lines and services.

9 Acknowledgements

We would like to thank Edison Klafke Fillus for his contribution in the experiment. This work was supported by CNPq: Productivity grant numbers: 303761/ 2009-1 and 306608/2009-0 and Universal Project grant numbers: 472510/2010-0 and 481397/2009-4).

References

[1] R. T. Alexander, J. M. Bieman, and A. A. Andrews. *Towards the Systematic Testing of Aspect-Oriented Programs*. Colorado State University, Technical Report, 2004.

[2] O. A. L. Lemos, A. M. R. Vincenzi, J. C. Maldonado, and P. C. Masiero. Control and data flow structural testing criteria for aspect-oriented programs. *Journal of Systems and Software*, 80:862–882, June 2007.

[3] J. Zhao. Data-flow based unit testing of aspect-oriented programs. In *Proceedings of the 27th Conference on Computer Software and Applications*, Washington, DC, 2003.

[4] H. Melton and E. Tempero. An empirical study of cycles among classes in Java. *Empirical Software Engineering*, 12:389–415, 2007.

[5] R. Ré and P. C. Masiero. Integration testing of aspect-oriented programs: a characterization study to evaluate how to minimize the number of stubs. In *Proceedings of the Brazilian Symposium on Software Engineering (SBES'2007)*, pages 411–426, October 2007.

[6] R. Galvan, A.R. Pozo, and S.R. Vergilio. Establishing Integration Test Orders for Aspect-Oriented Programs with an Evolutionary Strategy. In *Proceedings of the Latin-American Workshop on Aspect Oriented Software (LA-WASP'2010)*, 2010.

[7] R. Ré, O. A. L. Lemos, and P. C. Masiero. Minimizing stub creation during integration test of aspect-oriented programs. In *Proceedings of the 3rd Workshop on Testing Aspect-Oriented Program (WTAOP'2007)*, pages 1–6, Vancouver, British Columbia, Canada, March 2007.

[8] A. Abdurazik and J. Offutt. Coupling-based class integration and test order. In *Proceedings of the 2006 International Workshop on Automation of Software Test (AST'2006)*, pages 50–56, Shanghai, China, May 2006. ACM.

[9] L. C. Briand, J. Feng, and Y. Labiche. Using genetic algorithms and coupling measures to devise optimal integration test orders. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002)*, July 2002.

[10] L. C. Briand, J. Feng, and Y. Labiche. *Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders*. Carleton University, Technical Report SCE-02-03, October 2002.

[11] R.V. Cabral, A.R. Pozo, and S.R. Vergilio. A Pareto Ant Colony Algorithm Applied to the Class Integration and Test Order Problem. In *22nd IFIP International Conference on Testing Software and Systems (ICTSS'2010)*. Springer, 2010.

[12] A. Pozo, G. Bertoldi, J.C. Árias, R.V. Cabral, and S.R. Vergilio. Multi-objective optimization algorithms applied to the class integration and test order problem. *Software Tools for Technology Transfer*, 2011. , Submitted.

[13] W.K. Assunção, T.E. Colanzi, A.R.T Pozo, and S.R. Vergilio. Establishing integration test orders of classes with several coupling measures. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'2011)*, pages 1867–1874, 2011.

[14] T.E. Colanzi, W.K. Assunção, S.R. Vergilio, and A.R.T. Pozo. Generating integration test orders for aspect oriented software with multi-objective algorithms. In *Proceedings of the Latin-American Workshop on Aspect Oriented Software (LA-WASP'2011)*, 2011.

[15] T.E. Colanzi, W.K. Assunção, A.R.T Pozo, and S.R. Vergilio. Integration testing of classes and aspects with a multi-evolutioanry and coupling-based approach. In *Proceedings of the 3th International Symposium on Search Based Software Engineering (SSBSE'2011)*, pages 188–203. Springer Verlag, 2011.

[16] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture Notes in Computer Science*, pages 849–858, 2000.

[17] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[18] V. Pareto. *Manuel D'Economie Politique*. Ams Press, Paris, 1927.

[19] L. C. Briand and Y. Labiche. An investigation of graph-based class integration test order strategies. *IEEE Transactions on Software Engineering*, 29(7):594–607, July 2003.

[20] D. C. Kung, J. Gao, P. Hsia, J. Lin, and Y. Toyoshima. Class Firewall, test order and regression testing of Object-Oriented programs. *Journal of Object-Oriented Programming*, 8(2):51–65, 1995.

[21] Y. L. Traon, T. Jéron, J.-M. Jézéquel, and P. Morel. Efficient object-oriented integration and regression testing. *IEEE Transactions on Reliability*, pages 12–25, 2000.

[22] K.-C. Tai and F. J. Daniels. Test order for inter-class integration testing of object-oriented software. In *Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'1997)*, pages 602–607. IEEE Computer Society, August 1997.

[23] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. A test strategy for object-oriented programs. In *Proceedings of the 19th International Computer Software and Applications Conference (COMPSAC'1995)*, August 1995.

[24] R.E. Tarjan. Depth firstsearch and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[25] M. Ceccato, P. Tonella, and F. Ricca. Is AOP code easier or harder to test than OOP code. In *Proceedings of the 1st Workshop on Testing Aspect-Oriented Program (WTAOP'2005)*, Chicago, Illinois, 2005.

[26] M. Harman. The current state and future of search based software engineering. In *Proceedings of the Future of Software Engineering (FOSE'2007)*, pages 342–357. IEEE Computer Society, May 2007.

[27] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, January 1989.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182 –197, April 2002.

[29] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[30] ANNAS. Graph implementation and algorithm package. Online, 2011. Available at: http://code.google.com/p/annas/. Access: August, 2011.

[31] Joseph P. Bigus and Jennifer Bigus. *Constructing Intelligent Agents Using Java*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2001.

[32] J.J. Durillo, A.J. Nebro, and E. Alba. The jMetal framework for multi-objective optimization: Design and architecture. In *Proceedings of the Congress on Evolutionary Computation (CEC'2010)*, pages 4138–4325, Barcelona, Spain, July 2010.

[33] David A. van Veldhuizen and Gary B. Lamont. Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC'1999)*, pages 351–357, New York, NY, USA, 1999. ACM.

[34] J.L. Cochrane and M. Zeleny. *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[35] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.

[36] S. Gárcia, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[37] R-Project. The R project for statistical computing. <http://www.r-project.org/>, december 2010.