A Pareto Ant Colony Algorithm Applied to the Class Integration and Test Order Problem

Rafael da Veiga Cabral, Aurora Pozo, and Silvia Regina Vergilio *

Computer Science Department, Federal University of Paraná
CP: 19081, CEP 19031-970, Curitiba, Brazil
{rafaelv,aurora,silvia}@inf.ufpr.br

Abstract. In the context of Object-Oriented software, many works have investigated the Class Integration and Test Order (CITO) problem, proposing solutions to determine test orders for the integration test of the program classes. The existing approaches based on graphs can generate solutions that are sub-optimal, and do not consider the different factors and measures that can affect the stubbing process. To overcome this limitation, solutions based on Genetic Algorithms (GA) have presented promising results. However, the determination of a cost function, which is able to generate the best solutions, is not always a trivial task, mainly for complex systems with a great number of measures. Therefore, we introduce, in this paper, a multi-objective optimization approach to better represent the CITO problem. The approach generates a set of good solutions that achieve a balanced compromise between the different measures (objectives). It was implemented by a Pareto Ant Colony (P-ACO) algorithm, which is described in detail. The algorithm was used in a set of real programs and the obtained results are compared to the GA results. The results allow discussing the difference between single and multi-objective approaches especially for complex systems with a greater number of dependencies among the classes.

Keywords: Integration testing, object-oriented software, multi-objective, ant colony algorithm

1 Introduction

Software test is considered a fundamental activity to ensure software quality. It should be conducted in an incremental strategy [16]. In the context of Object Oriented (OO) software, this strategy includes different levels [10, 2]: method, class, cluster and system levels. In general terms, the base code (or component) should be developed, unit-tested, integrated and tested. A common problem in the integration phase is to determine the order in which classes are integrated and tested. This order is generally referred to the inter-class test order and is important because it affects [17]: the order in which classes are developed; the design of test cases; the number of created stubs for classes; the order in which inter-class faults are detected.

Sometimes, the class under test requires another class to be available before it can be executed. This kind of relationship is named dependency between classes, and can

* This work is partially supported by CNPq

require the creation of a stub to emulate the functionality that is used by the first class. The creation of stubs is an expensive and error-prone operation, and for that reason, the minimization of the number of stubs created during the integration testing is an important problem to be solved, called the Class Integration and Test Order (CITO) problem [1].

When there are no dependency cycles, the CITO problem can be solved by a simple reverse topological ordering of classes considering their dependency [12]. However, recent studies with real Java systems show that the presence of complex cycles is very common [13], and solutions to find an optimal order to minimize the stubbing effort are essential. Given such importance, we find in the literature, many works that address this subject. These works propose solutions based on graphs [17, 1, 12, 18, 6]. In general, the solution is obtained by removing, from the graph, dependencies that maximize the number of broken cycles. However, many times, these solutions are sub-optimal. Other disadvantage of the graph based approaches is that the cost to construct a stub may depend on many factors and can not be completely measured or estimated [4]. For example, number of attributes of a class, number of calls or distinct methods invoked, constraints related to organizational or contractual reasons, etc. To adapt the most graph-based solutions to consider these factors seems difficult or even impossible.

Due to the computational complexity, the CITO problem has been subject of the Search Based Software Engineering (SBSE), a recent research field that explores the use of meta-heuristic techniques to the Software Engineering problems [9]. A solution based on Genetic Algorithms (GA) was proposed in [4]. The authors use coupling measures to determine the stubbing complexity. The implemented GA is evaluated in real programs by using different four functions: number of broken dependencies, number of attributes, number of methods, and a geometric average of attributes and methods [3]. The obtained results are very promising when compared with the graph based solutions, which consider only the dependency. This solution allows considering different factors to establish the test orders. However, the choice of the more adequate evaluation function for the GA is not always a trivial task. The authors propose a procedure to find weights for all coupling measures, which is based on subjective steps and can be very expensive and labor-intensive for complex cases. This makes difficult the use of the solution based on GA in practice.

To overcome this limitation and to obtain solutions more adequate that consider the real constraints and diverse factors that may influence the CITO problem, we propose, in this paper, the use of a multi-objective search based approach. Such approach treats the CITO problem as a combinatorial constrained multi-objective optimization problem, more precisely, a problem where the goal is to find a set of test orders which satifies constraints and optimizes different factors.

To implement and evaluate the introduced approach, we can find in the literature many multi-objective algorithms. In this work, a multi-objective algorithm based on Ant Colony Optimization [8, 7] is explored. This algorithm is a meta-heuristic and has been successfully applied to solve many combinatorial optimization problems, such as, traveling salesman problem, sequential ordering problem, set covering problem, etc. To apply Ant Colony Optimization (ACO), the optimization problem is transformed into

the problem of finding the best path on a weighted graph. For that reason, we consider ACO one of the most suitable algorithm to the CITO problem.

The solutions returned by the multi-objective algorithm are evaluated according to Pareto dominance concepts [14] and represent a good trade off between the coupling measures: number of methods and attributes. To allow comparison with the GA approach, we conducted an experiment using the same benchmark used by Briand et al [3]. This benchmark is composed by real systems with varying complexities, what permits good insights about the use of both approaches. The results point out that the multi-objective approach presents a variety of good solutions even for complex cases.

The paper is organized as follows. Section 2 presents a review of works related to the CITO problem including the GA approach. Section 3 introduces our multi-objective approach and describes the implemented Ant Colony algorithm. Section 4 discusses the experimental results obtained. Section 5 concludes the paper and contains future research works.

2 The Class and Integration Test Order Problem

The integration test has the objective to detect faults associated to the interfaces between the modules when these are integrated to build the structure of the software, established in the project phase. It checks if the integrated system components work as desired. In the Object Oriented (OO) context the modules are classes that need to be integrated one at a time or, in some case in small groups. Inside this context, the CITO problem can be described as the identification of a priority order for integrating the classes. Once, an order of integration of the components has been assumed, this order can cause the implementation of components called stubs needed to simulate the behavior of tested and not yet integrated classes. The stubs represent additional costs to the project and its number must be reduced to the possible minimum. The minimization of the number of stubs created during the integration testing is an important problem to be solved, called the Class Integration and Test Order (CITO) problem [1].

Most approaches to the CITO problem are based on directed graphs, named ORD (Object Relation Diagrams), where the nodes represent classes and the edges represent their relationships. When there are no dependency cycles between the classes, the CITO problem can be solved by a simple reverse topological ordering of classes considering their dependencies [12]. However, this is not always the case, because most systems contain cycles. The approach proposed by Kung et al [12] was the first one to address this problem. The idea of this work is to identify strongly connected components (SCCs) in the graph and removing associations until no cycles remain. When there are more than one candidate associations for cycle breaking, a random selection is performed.

Tai and Daniels [17] define two class levels. Major-level numbers are assigned to classes based on inheritance and aggregation dependencies only. Then within each major level, minor-level numbers are assigned, based on association dependencies only. SCCs are identified in the major level and each edge of the SCCs receives a weight based on the related incoming and outgoing dependencies. Edges with higher weights are selected to break cycles because it is supposed to be related with more cycles. However, according to Briand et al [6] this assumption is not always true. There are cases

where class associations are not involved in cycles, and this solution is suboptimal in terms of the required number of test stubs.

In the work of Le Traon et al [18] the weights are assigned by the sum of incoming and outgoing frond dependencies for a given class within the SCC identified by Tarjan's algorithm. The frond dependency is a kind of edge, which is defined as going from a vertex (class) to one of its ancestors (a vertex that is traversed before it in a depth-first search that is the class depends on it, directly or indirectly). For each nontrivial SCC (with more than one vertex), the procedure above is then called recursively. The approach is non-deterministic because different sets of edges can be labeled as frond depending on the starting node, and when two or more nodes have the same weight, the selection is arbitrary.

A graph-based approach that combines the works of Le Traon et al and Tai and Daniels was proposed by Briand et al [6]. They also use Tarjan's algorithm to identify SCCs. The association edges in the SCCs are assigned with weights corresponding to the estimated number of involved cycles. This number is calculated according to the number of incoming and outgoing dependencies. The edge with highest weight is removed. The process is repeated until no SCC remains. The main advantages of this approach are that it does not break inheritance and aggregation edges, and computes the weights in a more precise way.

The works mentioned above present several limitations [4]. The most graph-based solutions consist in recursively identifying SCCs and in each SCC removing one dependency that maximizes the number of broken cycles. They optimize the decision without determining the consequences on the ultimate results. There are situations where breaking two dependencies has a lower cost than breaking only one that would make the graph acyclic in one step. Other disadvantage pointed out by Briand et al [4] is that the cost to construct a stub may be depend on many factors and can not be completely measured or estimated. For example, number of attributes of a class, number of calls or distinct methods invoked, constraints related to organizational or contractual reasons, and etc. To adapt the most graph-based solutions to consider these factors seems difficult or even impossible.

The work of Abdurazik and Offutt [1] consider more information in the minimization of the stubbing effort. The weights are derived from quantitative analysis of nine introduced kind of couplings, and are assigned to the edges and nodes. The coupling measures use number of parameters, number of return value types, number of variables and number of methods. The node weight is related to the estimated cost of removing it. If a class is used by multiple classes, then all or part of the same stub for that class may be shared among all classes that use it, thus reducing the cost of stubbing. The weight of a node is at least as high as the maximal weight of all incoming edges (assuming total sharing of the stub), and no higher than the sum of the weights of all incoming edges (assuming no sharing of the stub). The evaluation of this approach present positive results when compared with the approaches described before.

With this same objective, to allow the use of different kind of constraints and coupling measures, the approach based on Genetic Algorithms [4] has presented the most promising results. The authors used a fitness cost function based on different coupling measures. Number of attributes and methods necessary for the stubbing procedure are

considered besides the dependency factor. Briand et al [3, 5] conducted an experiment with a set of real programs and studied four different fitness functions: 1) number of dependencies; 2) method coupling; 3) attributes coupling; and 4) an aggregation of attribute and method coupling given by a geometric average. For each function the GA was executed 10 times. Thus, they have generated 40 solutions for each of the mentioned system. The solutions reported by Briand et al [3, 5] are presented in Table 1. For example, we can observe that for ATM System, the GA approach found the best solutions (highlighted entries in the table), with respectively 39 attributes and 13 methods, by using an average as fitness function.

Table 1. Solutions found by GA algorithm

ATM System										
Function	1	2	3	4	5	6	7	8	9	10
Dependencies	(52,19)	(54,19)	(67,13)	(67,19)	(52,19)	(46,19)	(46,19)	(45,19)	(59,19)	(47,13)
Attributes	**(39,13)**	(39,19)	**(39,13)**	(39,19)	(39,19)	(39,19)	**(39,13)**	**(39,13)**	(39,19)	**(39,13)**
Methods	**(39,13)**	(67,13)	(59,13)	(67,13)	(46,13)	(46,13)	(60,13)	(61,13)	**(39,13)**	(67,13)
Average	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**	**(39,13)**

ANT System										
	1	2	3	4	5	6	7	8	9	10
Dependencies	(187,26)	(187,26)	**(157,26)**	(187,26)	(187,26)	(213,22)	(187,26)	**(157,26)**	**(157,26)**	**(157,26)**
Attributes	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**	**(131,33)**
Methods	**(178,19)**	(184,19)	(184,19)	(197,22)	(227,22)	(227,22)	(197,22)	(229,22)	(226,22)	(197,22)
Average	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**	**(136,29)**

BCEL System										
Functions	1	2	3	4	5	6	7	8	9	10
Dependencies	(128,73)	(128,70)	(125,72)	(125,75)	(128,73)	(125,72)	(127,73)	(127,73)	(125,72)	(125,72)
Attributes	(47,86)	(47,87)	(47,85)	(46,84)	(46,85)	(46,84)	(46,76)	(46,83)	(46,85)	(46,84)
Methods	(131,67)	(125,70)	(131,70)	(131,70)	(131,67)	(134,69)	(133,69)	(134,69)	(138,70)	
Average	(56,70)	(55,72)	(58,72)	**(48,73)**	(54,73)	(53,73)	(59,73)	**(47,74)**	(59,73)	(50,74)

DNS System										
Functions	1	2	3	4	5	6	7	8	9	10
Dependencies	(22,11)	(28,11)	(28,11)	**(19,11)**	(22,11)	**(19,11)**	(22,11)	(28,11)	(28,11)	**(19,11)**
Attributes	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**
Methods	(28,11)	(28,11)	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	(22,11)	(28,11)	(28,11)	(22,11)
OCplx	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**	**(19,11)**

SPM System										
Functions	1	2	3	4	5	6	7	8	9	10
Dependencies	(149,28)	(149,28)	(149,28)	(149,28)	**(146,27)**	**(146,27)**	(149,28)	(149,28)	(149,28)	(149,28)
Attributes	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**	**(146,27)**
Methods	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**
Average	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**	**(148,26)**

The aggregation function is in fact a multiple objectives cost function, and is an alternative method to use a mono-objective algorithm to solve a multi-objective problem. However, Briand et al [4, 3] conclude that a practical issue when using a multiple objectives cost function is to determine appropriate weights. For that reason, Briand et al [4, 3] suggest a set of steps to select the best weights to be used. This set involves subjective aspects, such as the choice of minimal values for the measures. In complex cases the best solution can never be reached and the proposed procedure can be very difficult to apply for a great number of coupling measures or for complex cases. There-

fore, we introduce in next section a multi-objective approach, which obtains a set of good solutions and achieves a balanced compromise between the measures.

3 A Multi-Objective Approach

We observe in the last section that the CITO problem is a constrained multi-objective optimization problem as it may involve a number of different objectives; different coupling measures can be used as well as diverse factors that may influence the stubbing process. Therefore, we introduce, in this section, a multi-objective approach. We describe a representation for the solutions, objectives to be evaluated and implemented algorithm - a Multi-objective Ant Colony algorithm.

3.1 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 that better represent the possible compromise among them.

The general multi-objective maximization problem (with no restrictions) can be stated as to maximize Equation 1.

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

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 maximization problem, the solution $\overrightarrow{x}$ dominates $\overrightarrow{y}$ if:

$$\forall f_i \in \overrightarrow{f}, i = 1...Q, f_i(\overrightarrow{x}) \geq f_i(\overrightarrow{y}), and\ \exists f_i \in \overrightarrow{f}, f_i(\overrightarrow{x}) > f_i(\overrightarrow{y})$$

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

The goal is to discover solutions that are not dominated by any other in the objective space. A set of non-dominated objective vectors is called Pareto optimal and the set of all non-dominated vectors is called Pareto Front.

The Pareto optimal set is helpful for real problems, for example, engineering problems. It provides valuable information about the underlying problem [11]. In most applications, the search for the Pareto optimal is NP-hard [11], then the optimization problem focuses on finding an approximation set, as close as possible to the Pareto optimal.

3.2 Representing the CITO Problem as Multi-objective

An important issue in the implementation of a meta-heuristic algorithm is the chosen representation to describe the solutions of the problem. This choice will influence on the implementation of all the stages of the algorithm. In this case, the chosen representation for the problem is simple, with the solution being represented by a vector whose positions assume an integer number in the interval [1, N], being N the number of classes.

Thus, being each class represented by a number, an example of valid solution for a problem with 10 class would be (2,8,1,3,10,4,5,6,7,9). In this example, the first class to be tested and integrated would be the class represented by number '2'. A solution for the CITO problem is a permutation of the classes.

Other issue related to multi-objective algorithms is the choice of the objective functions. As mentioned before, many possible measures and factors can be used to the CITO problem: for example, coupling, cohesion, fault-proneness, contractual, process and time constraints, etc. Different meta-heuristics can be used. In this work, we chose an ACO algorithm and to allow comparison with the GA approach, we use two functions based on the same coupling measures used in the works of Briand et al [4, 3].

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

- $ACplx(o)$ (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 (the number of attributes locally declared in the target class when references/pointers to the target class appear in the argument list of some methods in the source class).This information is an input for the algorithm and is represented by a matrix $A(i, j)$, where rows and columns are classes and i depends on j. Then, for a given test order o and a set of d dependencies to be broken, the attribute complexity $ACplx$ is calculated according to Equation 2.

$$ACplx(o) = \sum_{i=1,n} \sum_{j=1,n} A(i, j); j \neq k \tag{2}$$

  Where n is the total number of classes and k is any class included before the class i, in test order o.

- $MCplx(o)$ (method complexity): The method complexity counts the number of methods that would have to be emulated in the stub if the dependency were broken (the number of methods locally declared in the target class which are invoked by the source class methods). This information is an input for the algorithm and is represented by a matrix $M(i, j)$, where rows and columns are classes and i depends on j. Then, for a given test order o and a set of d dependencies to be broken, the method complexity $MCplx$ is computed as defined by Equation 3.

$$MCplx(o) = \sum_{i=1,n} \sum_{j=1,n} M(i, j); j \neq k \tag{3}$$

  Where n is the total number of classes and k is any class included before the class i, in test order o.

- Constraints: In this work, following Briand et al [4, 3] work, Inheritance and Composition dependencies cannot be broken. This means the base/container classes must precede child/contained classes in any order test order o. The dependencies that cannot be broken are inputs for the algorithm, provided by a precedence table.

Based on the measures and constraints presented above, the problem is the search for an order that minimizes two objectives: the method and attribute complexities.

3.3 Multi-Objective Ant Colony Optimization Algorithm

The Ant Colony Optimization Algorithm (ACO) was introduced by [8]. ACO is inspired by the behavior of real ant colonies, in particular, by their foraging behavior. One of its main ideas is the indirect communication among the individuals of a colony or agents, called (artificial) ants, based on an analogy with trails of a chemical substance, called pheromone, which real ants use for communication. The (artificial) pheromone trails are a kind of distributed numeric information which is modified by the ants to reflect their experience accumulated while solving a particular problem.

The basic idea of ACO algorithms come from the ability of ants to find shortest paths from their nest to food locations. Considering a combinatorial optimization problem, an ant iteratively builds a solution. This constructive procedure is conducted using at each step a probability distribution, which corresponds to the pheromone trails in real ants. Once a solution is completed, pheromone trails are updated according to the quality of the best solution built. Hence, cooperation between ants is performed by a common structure that is the shared pheromone matrix. In addition to this, the algorithm discussed in this paper is based on the Pareto Ant Colony (P-ACO) algorithm, which is based on the Ant Colony System algorithm and was originally proposed to solve the Multiobjective Portfolio Selection problem [7].

P-ACO works with k pheromone matrices, where k is the number of objectives, and uses an aggregation heuristic function computed from the k objectives. The transition rule used to choose the next class j to be included in a test order o is given by Equation 4.

$$j = \begin{cases} \text{argmax}_{j \in U} \left[\sum_{k=1}^{K} p_k \cdot \tau_{ij}^k \right]^{\alpha} \cdot \eta_{ij}^{\beta} & \text{if } q \leq q_0 \\ p\left(j\right) & \text{otherwise} \end{cases} \tag{4}$$

where $p(j)$ is given by the probability, represented in Equation 5.

$$p(j) = \begin{cases} \dfrac{\left[\sum_{h=1}^{K} p_k \cdot \tau_{ij}^k \right]^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in U} \left[\sum_{h=1}^{K} p_k \cdot \tau_{ij}^k \right]^{\alpha}} & \text{if } j \in U \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where the pheromone matrices are represented by τ^k.

According to Pasia et al [15] this rule is a straightforward extension for multiple pheromone matrices of the rule used in the Ant Colony System [8]. The parameters α and β determine the relative influence of the pheromone and heuristic information, respectively; η_{ij}^k is the heuristic information of the objective k, $p^k \in [0, 1]$ are weights, which are uniformly distributed such that $\sum_{k=1}^{k} = 1$. In each iteration, a weight vector is assigned to each ant. The parameter q is a random number uniformly distributed in the interval [0, 1], and $q_0 \in [0, 1]$ is a parameter that defines the intensification and diversification properties of the algorithm.

The local pheromone update for all matrices is performed whenever an ant chooses a sequence of classes (i, j) and uses the following rule: $\tau_{ij}^k = (1 - \rho) \cdot \tau_{ij}^k + \rho \cdot \tau_0$, where ρ is the rate of evaporation and τ_0 is the initial pheromone value.

The global pheromone update is performed after all ants of the population have built a test order. For each objective k best and second-best solution is determined and then,

the following rule is used: $\tau_{ij}^k = (1-\rho)\cdot\tau_{ij}^k + \rho\cdot\Delta\tau_{ij}^k$, where $\Delta\tau_{ij}^k$ receives as value: 15 if the (i,j) component belongs to the best and second best solutions; 10 if (i,j) belongs only to the best path; 5 if it belongs just to the second best path; and 0 otherwise.

Algorithm 1 Pseudocode of P-ACO

```
/* Let C the number of classes */
/* Let N the number of iterations and Ants the number of ants */
F1, F2 are the objective functions
Initialize pheromone (F1, F2, τ0)
while nriter ≤ N do
  for all ant in Ants do
    p1 = rand (0, 1)
    p2 = 1 − p1
    TakeInitialCandidates = ()
    s = GenerateInitialPath ()
    BuildPath s = (s, q, q0, p1, p2, F1, F2)
    LocalPheromoneUpdate (s, F1, F2)
    s1 = LocalSearch (s, F1)
    s2 = LocalSearch (s, F2)
  end for
  for all objective k do
    Determine best b and second-best b'
    GlobalPheromoneUpdate (b, b', Fk)
  end for
  ParetoSetUpdate (P, s1, s2)
  nriter += 1
end while
```

Algorithm 1 describes the main steps of the P-ACO algorithm. First, an initial procedure is executed where the pheromone matrices are set to τ_0. The next step is an iterative loop, at each iteration, all the ants built a set, s, a test order solution.

The *Build_Path* procedure is used for each ant to build a test order. The ant chooses, based on a probabilistic decision (Equation 4), a class that has not yet been included in the test order s. The ant uses a candidate list composed by classes which do not have any precedence constraint, that is, either the class does not have any precedence by itself or all its precedence's classes had already been placed on the test order vector (precedence table). When the ant obtains a test order vector a local update is performed(`LocalPheronomeUpdate`).

After then, local searches are performed to achieve a strong exploitation of the search space. For each objective, a local search is performed on each ant (`LocalSearch`).

Finally, all ants are evaluated and a global pheromone update is performed (`GlobalPheromoneUpdate`) based on the best solutions of the iteration. The archive of the best solutions found so far (that is, the approximation of the Pareto front), is also updated (`ParetoSetUpdate`). At the end of the iterative loop, the solutions presented in the archive are the solutions obtained by the algorithm.

4 Experimental Results

The goal of this section is to describe the application of the P-ACO algorithm for the CITO problem in real systems and to compare its results with the GA approach. To do this, we used the same benchmark from the work of Briand et al [3]. This benchmark is composed of five real systems: ATM, ANT, SPM, BCEL and DNS. According to [3] these systems are deemed to be of sufficient size and of varying complexity. This allows a better evaluation considering different characteristics of the systems, which are described in Table 2. The systems ATM, ANT, SPM, and BCEL have class diagrams of reasonable (and comparable) sizes (between 19 and 45), but with very different numbers of cycles (from 30 for ATM to 416,091 for BCEL). On the other hand, the DNS system has the greatest number of classes and almost the same number of relationships of BCEL system, but the smallest number of cycles (fewer number than ATM, ANT, and SPM).

Table 2. Detailed Information about the Systems

System	Classes	Uses	Associations	Compositions	Inheritance	Cycles	LOC
ATM	21	39	9	15	4	30	1390
ANT	25	54	16	2	11	654	4093
SPM	19	24	34	10	4	1178	1198
BCEL	45	18	226	4	46	416,091	3033
DNS	61	211	23	12	30	16	6710

The P-ACO algorithm was executed with the following parameters: $q_0 = 0.4,\ \alpha = 1,\ \beta = 1,\ \tau_{min} = 0.0001$ and $\tau_0 = 1.0$. These values were got from the original P-ACO [7]. The number of ants is equal to the number of classes of each system, and the number of iterations is equal to twice this number except for the DNS system, where a lower number was used. These parameters are presented in Table 3. Observe that the population (number of ants) and iterations are lower than the values used by Briand et al [3]. The GA used a population size of 100 and a number of generation of 500.

As explained on Section 3.3, the P-ACO algorithm uses two local searches, on this implementation, a neighbor of 20 is explored and the stopping criterion is a number of 100 iterations without improvement on the best solution. The P-ACO algorithm was executed five times for each system. A performance evaluation between the proposed algorithm and the Briand et al [3] algorithm is not possible because their algorithm is not available. Then, this work tries to understand the benefit of one approach or the other based on the differences and similarities of the results.

4.1 Results and Analysis

The P-ACO results are presented for each system in Table 4. Each line presents the attribute and method complexities of the solutions for one independent run of the P-ACO

Table 3. Number of Ants and Iterations

System	ATM	ANT	SPM	BCEL	DNS
Ants	20	20	20	40	60
Iterations	40	40	40	80	80

algorithm. The non-dominated solutions, considering all the executions are highlighted. In this section, the obtained results are compared to the GA results presented in Table 1, considering the approximation of the Pareto front.

For ATM system the P-ACO algorithm found solutions with attribute complexity of 13 and method complexity of 39. These values are the best known approximation of the Pareto Front. For the GA algorithm, the same solutions are found when the used function is the average of the attribute and method complexities, as mentioned before. On the other hand, when the function uses only one complexity measure, the solutions are only good with respect to the metric employed.

In the ANT system, the P-ACO algorithm found solutions with better balance between the attribute and method complexities, forming the best approximation of the Pareto Front. The GA algorithm also found some of these solutions when the used function is the average of the attribute and method complexities. However, P-ACO has found two more solutions in the approximation of the Pareto Front. Again, when the GA algorithm uses a function based only on one complexity metric, the solutions are only good with respect to the metric employed, but the GA has more problems to find the best solutions and some runs found sub-optimal solutions.

For SPM, the approximation of the Pareto Front contains two non-dominated solutions with (146,27) and (148,26) respectively for attribute and method complexities. The P-ACO and GA algorithms found these solutions. But, the GA has found one solution when the attribute complexity function is used and the other one when the method complexity function is used. Moreover, these solutions were not found when the aggregated function is used.

For BCEL, note that there are only two solutions from the GA algorithm in the approximation of the Pareto Front. These solutions have (47,74) and (48,73) for attribute and method complexity respectively, and they were found by the function that aggregates both complexity measures. Besides these solutions, the P-ACO algorithm found another six solutions that are on the approximation of the Pareto Front. The BCEL System has the greatest number of solutions in the approximation of the Pareto Front, and this fact reveals a difference between the comparative study between GA and P-ACO algorithms.

The DNS System has the greatest number of classes, however, its complexity seems like to the ATM system since for all P-ACO executions only one solution was found with (19,11) as attribute and method complexities, respectively. These values are the best known approximation of the Pareto Front. For the GA algorithm, the same solutions are found when the used function is the aggregation of the attribute and method complexity. On the other hand, when the function uses only one complexity metric, the solutions are only good with respect to the metric employed.

Table 5 presents a comparison between the number of solutions found in the approximation of the Pareto Front by the compared algorithms. It is possible to observe that GA and P-ACO present similar behaviour for some systems. This behaviour can be explained because some systems have similar measures for attribute and method complexities, that is, when the attribute complexity grows the method complexity also grows. On the other hand, systems like BCEL exhibit different behaviour for attribute and method complexities. In these cases the P-ACO algorithm is the most suitable.

It is important to remark that the GA used four different complexity functions and the solutions were not always found by the same function. Consequently, some effort must be spent to determine the best function for the GA approach. It seems that this effort is greater for complex systems, with a great number of dependencies between classes, such as BCEL. This does not happen with the P-ACO approach.

Table 4. Solutions found by P-ACO algorithm

ATM System	
1	**(39,13)**
2	**(39,13)**
3	**(39,13)**
4	**(39,13)**
5	**(39,13)**

ANT System						
1	**(157,26)**	**(136,29)**	(184,19)	**(162,25)**	(183,22)	**(131,33)**
2	**(136,29)**	**(157,26)**	(183,22)	(168,25)	(184,19)	
3	**(157,26)**	**(136,29)**	(184,19)	(183,22)	(170,25)	**(131,33)**
4	**(157,26)**	**(136,29)**	**(178,19)**	**(163,22)**		
5	**(157,26)**	(162,25)	**(136,29)**	(183,22)	(184,19)	**(131,33)**

BCEL System									
1	(45,77)	(130,66)	(57,69)	**(78,68)**	(55,71)	(54,73)			
2	(79,69)	(128,68)	(129,67)	(45,77)	**(49,72)**	(131,66)	(54,71)		
3	(129,68)	(45,77)	(98,70)	**(46,75)**	(130,67)	(133,66)	(52,72)	(56,71)	(126,69)
4	(45,77)	**(130,66)**	**(54,69)**	(53,75)	(50,76)				
5	(105,68)	(57,70)	**(45,76)**	(54,71)	(134,66)	**(127,67)**			

DNS System	
1	**(19,11)**
2	**(19,11)**
3	**(19,11)**
4	**(19,11)**
5	**(19,11)**

SPM System		
1	**(146,27)**	**(148,26)**
2	**(146,27)**	**(148,26)**
3	**(148,26)**	**(146,27)**
4	**(146,27)**	**(148,26)**
5	**(148,26)**	**(146,27)**

Table 6 details the solutions found by P-ACO and GA algorithms for BCEL system. In this table it is possible to note different solutions that have in common a good trade off between the two complexity metrics. Remark that all these solutions are non dominated, hence no one can be considered to be better than any other with respect to both complexity metrics.

When the P-ACO approach is used, the tester can take advantage from the variety of these solutions according to his (or her) preferences (needs). He (or she) can conduct the integration test by prioritizing either attribute or method complexities. On the other hand, the tester can use other preference information about the classes, such as constraint related to contractual aspects to select an order from a larger range of good solutions than if the GA approach is used.

Table 5. Number of Solutions on the Approximation of the Pareto Front

Systems	ATM	ANT	SPM	BCEL	DNS
P-ACO	1	6	2	6	1
GA	1	4	2	2	1
Total	1	6	2	8	1

Table 6. Attribute and Method Complexities of BCEL system

Solutions	Algorithms	Attribute Complexity	Method Complexity
1	P-ACO	45	76
2	P-ACO	46	75
3	GA	47	74
4	GA	48	73
5	P-ACO	49	72
6	P-ACO	54	69
7	P-ACO	68	78
8	P-ACO	130	66

5 Conclusions

In this paper we introduce a new approach based on multi-objective optimization to the CITO problem. The approach uses a Pareto Ant Colony algorithm that was adapted to produce test orders that represent a good tradeoff between the number of attributes and methods in the stubbing process.

The algorithm was evaluated in a benchmark of five real programs and its performance was compared to the GA performance by considering the approximation of the Pareto front. In this evaluation we observe that the multi-objective is very advantageous because different factors can be considered and a set of good solutions that achieve a balanced compromise between the considered measures is obtained without human intervention. In addition to this, the approach is applicable and present better results in complex cases, when the system being tested contains a large number of dependency cycles.

It is possible to argue that the greater the number of solutions found in the approximation of the Pareto Front and their distributions, the greater the ways to satisfy integration test plans considering the real world needs. This fact points out that the P-ACO algorithm and the research in multi objective combinatorial problems are more suitable to solve software engineering complex problems, such as the CITO.

Now, we intend to conduct experiments with other multi-objective algorithms, such as Non-dominated Sorting Genetic Algorithm (NSGA-II). Besides this, other objectives can be included in the CITO problem, as mentioned in this text. The performance of the P-ACO with more than two objectives should be evaluated in further studies, with other benchmarks.

References

1. Abdurazik, A., Offutt, J.: Coupling-based class integration and test order. In: International Workshop on Automation of Software Test. ACM, Shanghai, China (May 2006)
2. Binder, R.V.: Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley (2000)
3. Briand, L.C., Feng, J., Labiche, Y.: Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders. Carleton University, Technical Report SCE-02-03 (October 2002)
4. Briand, L.C., Feng, J., Labiche, Y.: Using genetic algorithms and coupling measures to devise optimal integration test orders. In: 14th International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy (July 2002)
5. Briand, L.C., Feng, J., Labiche, Y.: Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders. In: Proceedings of Software Engineeing with Computational Intelligence. pp. 204–234. Kluwer Academic Publishers (2003)
6. Briand, L.C., Labiche, Y.: An investigation of graph-based class integration test order strategies. IEEE Transactions on Software Engineering 29(7), 594–607 (July 2003)
7. Doerner, K., Gutjahr, W.J., Hartl, R.F., Strauss, C., Stummer, C.: Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. Annals of Operation Research (131), 79–99 (2004)
8. Dorigom, M., Socha, K.: An Introduction to Ant Colony Optimization. No. TR/IRIDIA/2006-010., Technical Report - IRIDIA (April 2006)
9. Harman, M.: The current state and future of search based software engineering. In: Proceedings of International Conference on Software Engineering / Future of Software Engineering 2007 (ICSE/FOSE '07). pp. 342–357. IEEE Computer Society, Minneapolis, Minnesota, USA (20-26 May 2007)
10. Harrold, M.J., McGregor, J.D., Fitzpatrick, K.J.: Incremental testing of object-oriented class structures. In: 14th International Conference on Software Engineering. pp. 68–80. IEEE Computer Society, Melbourne, Australia (May 1992)
11. Knowles, J., Thiele, L., Zitzler, E.: A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland (February 2006)
12. Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C.: A test strategy for object-oriented programs. In: 19th International Computer Software and Applications Conference. IEEE Computer Society (August 1995)
13. Melton, H., Tempero, E.: An empirical study of cycles among classes in Java. Empirical Software Engineering 12, 389–415 (2007)
14. Pareto, V.: Manuel D'Economie Politique. Ams Press, Paris (1927)
15. Pasia, J.M., Hart, R., Doerner, K.F.: Solving a bi-objective flowshop scheduling problem by Pareto-ant colony optimization. Lecture Notes in Computer Science pp. 294–305 (August 2006)
16. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw-Hill (2006)
17. Tai, K.C., Daniels, F.J.: Test order for inter-class integration testing of object-oriented software. In: 21st International Computer Software and Applications Conference. pp. 602–607. IEEE Computer Society (August 1997)
18. Traon, Y.L., Jéron, T., Jézéquel, J.M., Morel, P.: Efficient object-oriented integration and regression testing. IEEE Transactions on Reliability pp. 12–25 (2000)