Grammatical Evolution for the Multi-Objective Integration and Test Order Problem

Thainá Mariani, Giovani Guizzo, Silvia R. Vergilio and Aurora T. R. Pozo
Computer Science Department
Federal University of Paraná
Curitiba, Paraná, Brazil
{tmariani, gguizzo, silvia, aurora}@inf.ufpr.br

ABSTRACT

Search techniques have been successfully applied for solving different software testing problems. However, choosing, implementing and configuring a search technique can be hard tasks. To reduce efforts spent in such tasks, this paper presents an offline hyper-heuristic named GEMOITO, based on Grammatical Evolution (GE). The goal is to automatically generate a Multi-Objective Evolutionary Algorithm (MOEA) to solve the Integration and Test Order (ITO) problem. The MOEAs are distinguished by components and parameters values, described by a grammar. The proposed hyper-heuristic is compared to conventional MOEAs and to a selection hyper-heuristic used in related work. Results show that GEMOITO can generate MOEAs that are statistically better or equivalent to the compared algorithms.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

search based software engineering, multi-objective, grammatical evolution, hyper-heuristic, evolutionary algorithm

1. INTRODUCTION

In the Search-Based Software Engineering (SBSE) field, search based optimization techniques are used to automate the search for optimal or near-optimal solutions to software engineering problems. We can find SBSE approaches for solving problems related to many software engineering tasks, such as requirements, design, maintenance and testing [17]. This last task has received many attention and several testing problems have been successfully solved in the Search Based Software Testing (SBST) [31].

Among such problems, we can mention the ITO (Integration and Testing Order) problem [1, 5], a hard software testing problem that consists in finding a sequence of units to be tested, such that the stubbing cost is minimized. A unit is the smallest part of a software (procedure, class, method, aspect) that will be tested eventually. The problem starts when a unit A under test requires a unit B for a given functionality, but B is not yet implemented. What the tester must do in this situation is to develop a stub for B, which increases the testing cost. A stub is an emulation of a unit that is later discarded when such a unit is developed, thus it might be considered a waste of resources to develop unnecessary stubs. If the tester is smart enough, he/she will develop unit B first and then unit A (sequence of {B, A}), thus no stubs are required. However, in large systems it is not that simple since there are a lot of units and potentially several cyclic that cannot be broken without a stub. The idea is to reduce the testing cost by minimizing the resources employed into developing stubs. There are several objective functions for evaluating the stubbing cost of the solutions (orders of units). For instance, the number of required stubs, methods of stubs, attributes of stubs, classes, interfaces and so on [1]. In this sense, approaches based on Multi-Objective and Evolutionary algorithms (MOEAs) are the most promising. By the way, surveys on SBSE [17] show that MOEAs are the most used and preferred.

Nevertheless, the use of MOEAs involves many decisions [13]. Firstly, it is necessary to choose the algorithm to be used. Then, each parameter value must be chosen from a lot of possible values. Furthermore, the choice may depend on the problem representation and particularities. For instance, in a simple genetic algorithm, the software engineer needs to configure the population size, crossover and mutation operator, and respective probabilities. Therefore, making a decision taking into account all these aspects is a hard task that can be considered an optimization problem by itself [13]. Hyper-heuristics [7] are heuristics to select or generate heuristics for solving hard search problems, thus they can be used to select or generate MOEAs. A learning mechanism can be used by the hyper-heuristic for updating the heuristic preference based on its historical performance. The learning can be online, which is performed while the problem is being solved. The learning can also be offline, when it is first performed in a set of training instances and then the resulting heuristic is used to solve other instances. Hence, hyper-heuristics can be classified according to the nature of the heuristic search space (selection or generation) and the source of feedback (online, offline or no-learning) [7].

In the SBSE context, hyper-heuristics can contribute to obtain a holistic and generic SBSE [16], and have already

© 2016 ACM. ISBN .
DOI:

been explored by some works [2, 15, 18, 19, 21], including in the context of SBST [15, 18, 19]. Guizzo et al. [15] mention that the application of hyper-heuristics to the ITO problem is very suitable. There are many operators that can be used, given the permutation representation of the problem (each gene is the ID of a unit). In addition to this, experiments [1] show that no algorithm has been proved to be the best to solve ITO in different problem instances and contexts.

The work of Guizzo et al. [15] proposes a hyper-heuristic to solve the ITO problem. It automatically selects the best low-level heuristic that is a combination of crossover and mutation operators generally used by Evolutionary Algorithms (EAs). Nevertheless, the proposed hyper-heuristic presents a limitation: the tester still has to choose a specific algorithm, select the other components and tune some parameter values. Moreover, the hyper-heuristic is online, resulting in resources being allocated for the training mechanism during the problem solving. A good way to avoid this is to train the algorithm before solving the problem and reuse the trained heuristic.

To overcome this limitation, we propose an offline hyper-heuristic to generate MOEAs for solving the ITO problem. The generated MOEAs are distinguished by many components and parameters, including initialization, selection, mating, operators, replacement and archiving. We chose the generation of MOEAs because, as mentioned before, they are multi-objective algorithms and because existing MOEAs have already been explored for this problem. That way, we believe that a new MOEA, trained specifically for the ITO problem, can obtain good results. Furthermore, MOEAs have a great number of components and parameters that can be explored by the hyper-heuristic, resulting in many MOEAs possibilities. In addition, some works in the literature already used the generation of EAs in other contexts [3, 4, 25], presenting promising results.

Our hyper-heuristic is based on Grammatical Evolution (GE) [32]. It is called GEMOITO (Grammatical Evolution hyper-heuristic for the Multi-objective Integration and Test Order problem). GE is a type of Genetic Programming (GP) [20] that uses a grammar to generate programs. The grammar defines rules to be used in the GE evolutionary process. There are several works in the literature using GE to generate search algorithms [8, 24–26, 28–30], few of them are related to the generation of EAs [24–26]. Results obtained by these works encouraged us to use GE as the heuristic for generating MOEAs. The GE grammar is composed by several values for the components and parameters of MOEAs, thus the GE algorithm can generate a MOEA containing the best combination of components and parameters values.

GEMOITO is experimentally evaluated and compared with the MOEAs used in the literature: Nondominated Sorting Genetic Algorithm II (NSGA-II) [10] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [34], and with the online hyper-heuristic proposed by Guizzo et al. [15]. The obtained Pareto fronts were evaluated using the hypervolume indicator [35], the Kruskal-Wallis statistical test [11] and the Effect Size statistical test [9]. GEMOITO results are statistically equivalent or better than the compared hyper-heuristic, NSGA-II and SPEA2 results.

This paper is organized as follows. Section 2 summarizes related work. Section 3 presents GEMOITO, describing the grammar, defined components and parameters values. Section 4 describes the empirical evaluation: the research questions, the performed experiments and the obtained results. Finally, Section 5 contains conclusions and some future research works.

2. RELATED WORK

Hyper-heuristics have been explored by many works in the literature [6]. However, we find few works addressing hyper-heuristics and SBSE. They are presented next.

Basgalupp et al. [2] proposed a hyper-heuristic to generate an algorithm for the creation of effort-prediction decision trees. The algorithm is generated based on existing heuristic blocks. Kumari et al. [21] proposed a hyper-heuristic based on multi-objective genetic algorithm to solve the software module clustering problem. During the execution of the genetic algorithm, the hyper-heuristic selects a low-level heuristic to be applied. Twelve low-level heuristics are available, which are combinations of different selection, crossover and mutation operators. Jia et al. [19] investigate a simulated annealing hyper-heuristic that performs online learning in order to dynamically apply the best Combinatorial Interaction Testing (CIT) strategy in the testing activity. Jia [18] describes a selection hyper-heuristic framework for Search Based Software Testing (SBST). A set of meta-heuristics can be used to provide a global framework for the hyper-heuristic. The hyper-heuristic selects the heuristics to be applied by the meta-heuristic, which are changed based on the problem being solved. No experimentation was performed and just an idea of the framework is presented.

Guizzo et al. [15] propose HITO (A Hyper-heuristic for the Integration and Test Order Problem), an online selection hyper-heuristic. This is the work most related to ours. HITO selects, at each mating of a MOEA algorithm, the best low-level heuristic to be applied. In this context, a low-level heuristic is a combination of a crossover and a mutation operator, which are specific for the permutation representation of the problem. HITO implements two selection functions: Choice Function (CF) [27] and Multi-Armed Bandit (MAB) [22]. A quality measure is used to assess the performance of each low-level heuristic, taking into account the dominance concept and the number of matings of the algorithm. HITO was implemented using NSGA-II and evaluated in seven instances of the ITO problem, where it outperformed the ones obtained by some conventional MOEAs.

There are some works in the literature [8, 24–26, 28–30] using GE for the generation of search algorithms. Lourenço et al. [24–26] focus on the automatic generation and tuning of mono-objective EAs. However, we have not found works addressing the generation of multi-objective algorithms using GE. Some works address the automatic design of multi-objective algorithms using other techniques [3, 4, 12, 23, 33]. In the MOEA context, most works are related to the parameter tuning [12, 33]. The generation of MOEAs is explored by Bezerra et al. [3, 4], that use Iterated Racing algorithms and combine different MOEAs components, such as replacement, archiving and fitness assignment. None of these works generate algorithms for solving software engineering problems. As far as we know, our approach is the first one that uses a hyper-heuristic based on GE for solving a multi-objective software engineering problem.

The results of the presented works encouraged us to propose GEMOITO. We believe that changing MOEA components and parameters is an effective way to obtain specialized algorithms that can outperform generic ones in the ITO problem. The next section presents this hyper-heuristic and how the MOEA generation is done.

3. OUR HYPER-HEURISTIC

Grammatical Evolution hyper-heuristic for the Multi-Objective Integration and Test Order Problem (GEMOITO) is an offline generation hyper-heuristic, based on GE, to automatically generate MOEAs to solve the ITO problem. An offline hyper-heuristic uses a feedback mechanism for the training of the low-level heuristics before solving the problem, as opposed to online hyper-heuristics that employ the training during the problem solving (dynamically) [7]. An advantage of such hyper-heuristics is that, even though they usually need a great amount of resources to train the low-level heuristics, no extra resource is wasted during the problem solving. Moreover, the trained heuristics are expected to be reusable throughout other problem instances. Generation hyper-heuristics are used to generate low-level heuristics, instead of selecting existing ones as selection hyper-heuristics do [7]. The generated heuristics are expected to perform well on unknown instances of the problem, thus usually some sort of training is employed to find such robust heuristics.

GE [32] algorithms are a type of GP [20] capable of supporting such hyper-heuristics, given their main purpose of generating programs (that can be heuristics). GE uses a grammar to guide its evolutionary search. This grammar contains several rules, each one containing several possibilities. In the GP terminology, a rule is a non-terminal node, whereas each of its options can be a terminal or a non-terminal node. What the GE algorithm does is to map the chromosome into a tree using such a grammar. The chromosome is usually an integer vector, where each gene is mapped into an option (terminal or non-terminal) of one grammar rule. The algorithm keeps decoding genes into rules until a full solution is built. At the end, the tree is transformed into a phenotypic representation according to the mapper. The grammar is the most important artifact of a GE algorithm, since it dictates the possible nodes for the solution.

The choice of using GE to generate MOEAs was mainly inspired by [24–26], where the idea of generating EAs using GE obtained better results when compared with traditional EAs. However, when dealing with MOEAs, there are many components and parameters that can be explored. Bezerra et al. [3, 4] did this and obtained good results when compared with conventional MOEAs, but without using GE. Based on their works, we modeled these components and parameters in the GEMOITO grammar. In addition, we implemented the selection of other ones, such as population initialization, source of parent selection, type of replacement and more. Furthermore, we also implemented a widest range of possible values for them. Such a grammar is presented in Figure 1. Each item between "⟨" and "⟩" is a rule (non-terminal node), everything after "::=" represents its options, "|" divides the possible options to fulfill this rule, values without "⟨" and "⟩" are terminal nodes and λ is a `null` option.

```
⟨GA⟩ ::= ⟨populationSize⟩ ⟨initialization⟩ ⟨selection⟩ ⟨mating⟩
      ⟨replacement⟩ ⟨archive⟩

⟨populationSize⟩ ::= 50 | 100 | 150 | 200 | 250 | 300

⟨initialization⟩ ::= Random | Parallel Diversification

⟨selection⟩ ::= ⟨selectionOperator⟩ ⟨source⟩ ⟨fitnessAssignment⟩

⟨selectionOperator⟩ ::= K Tournament ⟨tournamentSize⟩ | Random
   | Roulette Wheel | Ranking

⟨tournamentSize⟩ ::= 2 | 4 | 6 | 8 | 10

⟨source⟩ ::= Population | Archive and Population

⟨fitnessAssignment⟩ ::= ⟨convergenceStrategy⟩ ⟨diversityStrategy⟩

⟨convergenceStrategy⟩ ::= λ | Dominance Rank | Dominance Strength
      | Dominance Depth | Raw Fitness

⟨diversityStrategy⟩ ::= λ | Crowding Distance
      | K-th Nearest Neighbor | Adaptive Grid
      | Hypervolume Contribution

⟨mating⟩ ::= ⟨matingOperators⟩ ⟨matingStrategy⟩

⟨matingOperators⟩ ::= ⟨crossoverOperator⟩ ⟨crossoverProbability⟩
      ⟨mutationOperator⟩ ⟨mutationProbability⟩

⟨crossoverOperator⟩ ::= λ | Two Points Crossover
      | Single Point Crossover | PMX Crossover | Cycle Crossover

⟨crossoverProbability⟩ ::= 1.0 | 0.95 | 0.9 | 0.8 | 0.5

⟨mutationOperator⟩ ::= λ | Swap Mutation | Insert Mutation
      | Scramble Mutation | Inversion Mutation

⟨mutationProbability⟩ ::= 0.01 | 0.02 | 0.05 | 0.1 | 0.2
      | 0.5 | 0.7 | 0.8 | 0.9 | 1.0

⟨matingStrategy⟩ ::= Steady State | Generational Two Children
      | Generational One Child

⟨replacement⟩ ::= Generational ⟨elitismSize⟩ ⟨fitnessAssignment⟩
      | Ranking ⟨fitnessAssignment⟩

⟨elitismSize⟩ ::= 0 | N * 0.01 | N * 0.05 | N * 0.1 | N * 0.5

⟨archive⟩ ::= Ranking ⟨fitnessAssignment⟩ ⟨archiveSize⟩

⟨archiveSize⟩ ::= 0 | N | N * 1.5 | N * 2
```

Figure 1: Grammar used by GEMOITO

The grammar is used to generate different MOEAs, but their template is always the same as shown in Algorithm 1. The rules of the grammar are all components or parameters usually present in MOEAs and each terminal node is a value for the parameter or an implementation for the component. For instance, ⟨*populationSize*⟩ denotes the population size parameter and everything after "::=" are values that can be used for this parameter. Similarly, ⟨*crossoverOperator*⟩ is a rule representing the crossover operator of the MOEA that can be none (λ), "Two Points Crossover", "Single Point Crossover", "PMX Crossover" or "Cycle Crossover" [14]. Note that, because ITO is a permutation problem, all the crossover and mutation operators available in the grammar are for permutation solutions. The first rule of the grammar (⟨*GA*⟩) defines the rules that can be used by the GE algorithm to generate a MOEA.

What the GE mapper does is to select the rule options according to the genotype and to replace the abstract operation in the MOEA template with a concrete value (parameter value or implementation for the component). At the end, the mapper returns a fully constructed and configured MOEA. GEMOITO receives this MOEA and executes

Algorithm 1: Template of the generated MOEAs

```
begin
    $population$ ← Population initialization;
    Evaluate ($population$);
    Archive ($population$);
    while $stop\ criteria\ is\ not\ achieved$ do
        $matingPopulation$ ← Selection ($population$);
        $offspringPopulation$ ← Crossover
          ($matingPopulation$);
        $offspringPopulation$ ← Mutation
          ($offspringPopulation$);
        Evaluate ($offspringPopulation$);
        Replacement ($offspringPopulation$, $population$);
        Archive ($offspringPopulation$);
```

it by using a training instance of the problem. The resulting Pareto front is then evaluated using hypervolume [35] to assess the performance of the MOEA. The hypervolume result is the "fitness" of the MOEA, which is then used, as in other Evolutionary Algorithms (EAs), to determine the parents, surviving solutions (MOEAs) and so on. During the GE search process, several MOEAs are generated using the grammar, but at the end only the best MOEA is given as result to solve the ITO problem.

One important thing to note is that when dealing with multiple objectives, there might be incomparable solutions, as known as non-dominated ones. However, these solutions are still compared by the algorithm during decision points. For example, the algorithm must decide which of two non-dominated solutions will survive for the next generation. In this sense, the MOEAs might employ some sort of fitness assignment strategy to assess the quality of solutions in a multi-objective environment. For instance, NSGA-II [10] uses Dominance Depth and Crowding Distance, whereas SPEA2 [34] uses Raw Fitness and K-th Nearest Neighbor. GEMOITO's grammar has rules for fitness assignment strategies for three procedures of the evolutionary process such as done in [3]: i) selection; ii) replacement; and iii) archiving. These rules encompass both convergence and diversity strategies. Therefore, GEMOITO lets the generated MOEAs evaluate each solution in three different ways (one for each procedure), focusing more on convergence or diversity, depending on the selected strategy. We expect to obtain more robust MOEAs using such an approach.

4. EMPIRICAL EVALUATION

In order to evaluate GEMOITO, we conducted experiments performing comparisons with existing approaches. These experiments are guided by two research questions:

RQ1: Can the MOEAs generated by GEMOITO outperform MOEAs used in the literature? To answer this question, we compared the MOEAs generated by GEMOITO with two conventional MOEAs used by [1]: Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [10] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [34].

RQ2: How do GEMOITO results compare to HITO? HITO is the hyper-heuristic proposed by Guizzo et al. [15] that also solves the same problem. HITO is an online selection hyper-heuristic, which contrasts with GEMOITO, an offline generation hyper-heuristic. Therefore, we can observe the benefits and disadvantages of both kinds of hyper-heuristics.

We used 7 real world systems for this evaluation: MyBatis (v3), AJHSQLDB (v18), AJHotDraw (v0.4), BCEL (v5.0), JHotDraw (v7.5.1), HealthWatcher (v9) and JBoss (v6.0.0M5). These are the same systems used in related work [1, 15] for which the data sets were provided by the authors of [1]. They vary in the implemented paradigm (Object Oriented – OO, or Aspect Oriented – AO), number of units, number of dependencies and lines of code.

The evaluation is divided in two phases: i) training (Subsection 4.1); and ii) testing (Subsection 4.2). In the former, we executed GEMOITO 10 times in order to generate 10 MOEAs. We also used GEMOITO to automatically tune the conventional MOEAs in order to obtain a fair comparison between them. In the testing phase, the generated algorithms, the tuned conventional MOEAs and HITO were executed in the 7 systems in order to test their performance. Similarly to related work and to allow comparisons, the generated MOEAs use two objective functions: number of operations (O) and number of attributes (A) [1, 15]. These two functions can be used in object-oriented and aspect-oriented software to measure the number of required operations and number of attributes to be emulated. The results were evaluated using the obtained Pareto fronts, the hypervolume indicator [35], the Kruskal-Wallis statistical test [11] and the Cohen's d effect size [9].

4.1 Training

We fixed the parameters of GEMOITO after a preliminary experimentation using similar values to the work of Lourenco et al. [25]. Table 1 presents these parameters.

Table 1: GEMOITO parameters

Parameter	Value
Population Size	100
Number of GE Fitness Evaluations	10.000
Number of Training Fitness Evaluations	2.000
Crossover Operator	Single Point Crossover
Crossover Probability	90%
Mutation Operator	Integer Mutation
Mutation Probability	1%
Selection Operator	Binary Tournament
Pruning Operator Probability	1%
Duplication Operator Probability	1%
Minimum of Genes in the Init. Pop.	10
Maximum of Genes in the Init. Pop.	20
Replacement Strategy	Ranking

Table 2: Components and parameters of ALG_6

Population size	50
Initialization	Random
Selection Op.	Ranking Converg. Strategy: Dominance Strength Divers. Strategy: K-th Nearest Neighbor
Source	Archive and Population
Mating Strategy	Steady State
Crossover Op.	-
Mutation Op.	Swap Mutation (100%)
Replacement	Generational Elitism Size: 5 Converg. Strategy: Raw Fitness Divers. Strategy: K-th Nearest Neighbor
Archive	Ranking Converg. Strategy: Raw Fitness Divers. Strategy: Hypervolume Contribution
Archive size	75

Table 3: Parameters of HITO, NSGA-II and SPEA2

Parameter	HITO	NSGA-II	SPEA2
Population Size	300	50	50
Maximum Fitness Evaluations	60,000	60,000	60,000
Crossover Operator	All	PMX	PMX
Crossover Probability	-	100%	95%
Mutation Operator	All	Swap	Swap
Mutation Probability	-	1%	5%
Archive Size	-	-	50
CF α	1.0	-	-
CF β	0.00005	-	-

With these parameters, GEMOITO was executed 10 times using the largest instance (AJHSQLDB), resulting in 10 different MOEAs named ALG_0 to ALG_9. Each MOEA was trained with a budget of 2,000 fitness evaluations. The best algorithm found is ALG_6. Its components and parameters are presented in Table 2. Due to space restrictions, we omitted the other algorithms, but we observed only few changes from one algorithm to another. For instance, ALG_3 differs from ALG_6 by using the Cycle Crossover operator with 80% probability, no elitism, no convergence strategy for archiving and using Crowding Distance rather than K-th Nearest Neighbor as diversity for selection.

Similarly, we used GEMOITO with slightly different grammars for automatically tuning both NSGA-II and SPEA2. The HITO parameter configuration is the same as presented in [15], however, with the addition in its dynamic selection of the crossover and mutation operators available in our grammar (Figure 1). Moreover, we used only the version of HITO with NSGA-II and CF, since it obtained the best results [15]. The parameters of HITO, NSGA-II and SPEA2 are presented in Table 3. HITO does not use crossover and mutation probabilities because it dynamically applies the operators according to the search stage. Furthermore, HITO has more individuals in the population when compared to the other algorithms, since it is online and because of that, requires more solutions for the training.

4.2 Testing

In the testing phase, first we executed all 13 algorithms (NSGA-II, SPEA2, HITO and the 10 MOEAs generated by GEMOITO) in the 7 real world systems for 30 independent runs using 60,000 fitness evaluations each run. As a result, each algorithm generated one Pareto front for each system in each independent run. At the end, each algorithm had its 30 obtained fronts merged, excluding repeated and dominated solutions. Therefore, each merged Pareto front contains all the non-dominated solutions found by an algorithm for a given system. This front is the best known Pareto front of the algorithm for that system (PF_{known}). The true Pareto fronts are not known for these problems.

The PF_{known} fronts can be seen in Figures 2-6. The fronts of HealthWatcher and JBoss were omitted, because they had only one solution. These are the smallest systems used in this work, thus have unit orders easy to find.

The fronts found by NSGA-II, SPEA2 and HITO for AJHSQLDB (Figure 3) are dominated by at least one algorithm generated by GEMOITO, but it is rather an unfair comparison since the algorithms were trained with this instance. However, although not entirely visible due to the great number of solutions, the same thing happens for other instances, such as MyBatis (Figure 2) and AJHotDraw (Figure 4). We

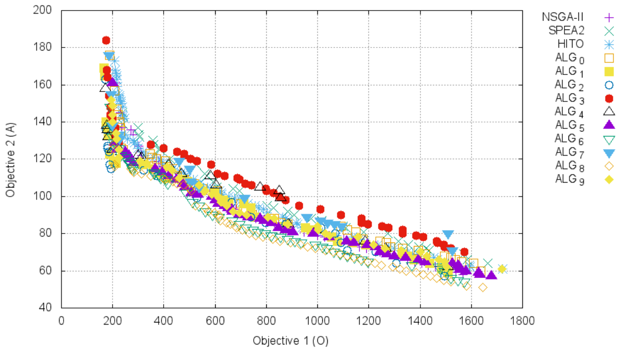

Figure 2: PF_{knwon} fronts found for MyBatis

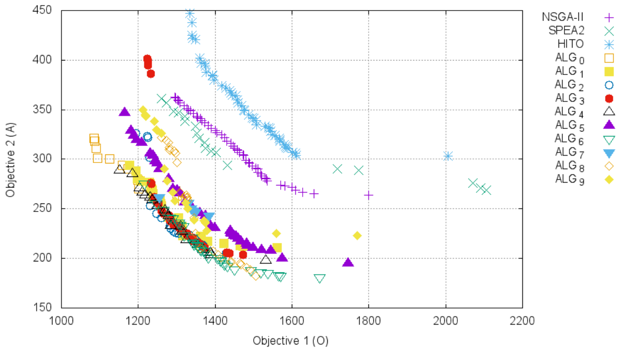

Figure 3: PF_{knwon} fronts found for AJHSQLDB

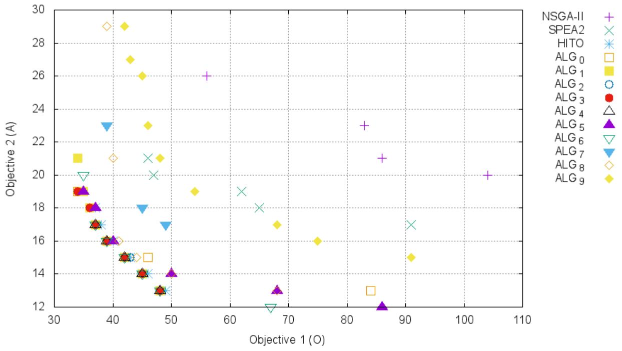

Figure 4: PF_{knwon} fronts found for AJHotDraw

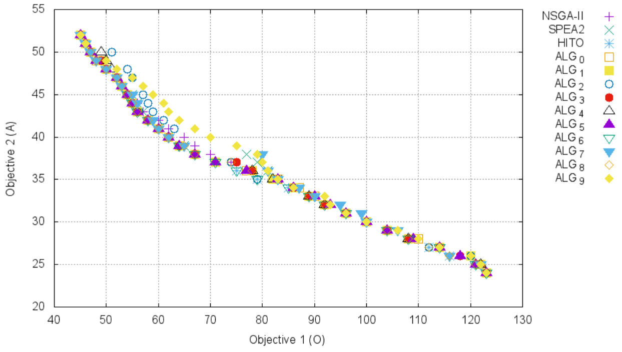

Figure 5: PF_{knwon} fronts found for BCEL

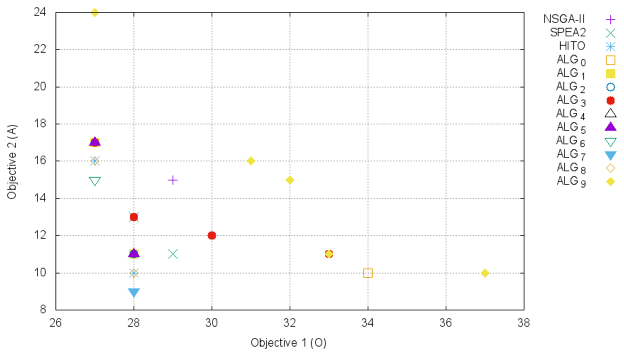

Figure 6: PF_{knwon} fronts found for JHotDraw

observed that the fronts of HITO and the generated algorithms (mostly ALG_6) remain relatively close when compared to the fronts of NSGA-II and SPEA2. Thus, we can state that both hyper-heuristics found the best results.

Since most fronts are mixed in the objective space, we also used the hypervolume indicator [35] to calculate the quality of the fronts of each algorithm. This indicator was chosen due to its capability of evaluating both the convergence and diversity. Besides, it does not need a true Pareto front for the assessment. Table 4 shows the hypervolume averages found by each algorithm in each system. These values were normalized using the worst possible point for the respective results. Moreover, values in bold represent the best values, or equivalent to the best ones, according to the Kruskal-Wallis statistical test with 95% of confidence.

As seen in Table 4, the conventional MOEAs NSGA-II and SPEA2 could only find statistically equivalent values for 2 systems. Still, these are the smallest systems and almost all algorithms were able to find values close to 1 (best possible value). For the bigger problems, HITO and the generated algorithms have the best results. Furthermore, for all systems, at least one algorithm generated by GEMOITO obtained greater hypervolume averages than both NSGA-II and SPEA2. The HITO results are similar as obtained in [15] regarding the comparison with the conventional MOEAs.

Comparing HITO and GEMOITO, for 2 systems (out of 7), HITO was not able to obtain equivalent results to the best one, which also happened to some generated algorithms (ALG_1, ALG_2, ALG_5 and ALG_8). In addition, some generated algorithms obtained results not so good as HITO did, e.g., ALG_0, ALG_3, ALG_4, ALG_7 and ALG_9. Notably, ALG_6 always obtained the best results, or results equivalent to the best ones. Summing it up, 5, out 10 algorithms generated by GEMOITO, presented worse results than HITO in terms of number of best or equivalent fronts. 4 of the generated MOEAs presented equal results and ALG_6 outperformed or equaled all the other algorithms.

As another source of comparison, we executed the Cohen's d [9] effect size. This statistical test gives the difference magnitude between two groups of values. In our work, the groups are the algorithms and each group has a set of 30 hypervolume values, one for each front obtained with the 30 independent runs. Table 5 presents the Effect Size results regarding each binary comparison. Negative values mean that the left algorithm (in the column header) performed worse than the right algorithm.

The Effect Size results show something similar as we observed using the Kruskal-Wallis test, with some slight differences since the effect size calculation is done in pairs of groups rather than using all groups at once. For instance, the Kruskal-Wallis test showed equality between HITO and ALG_6 for MyBatis, BCEL and JHotDraw. However, the effect size test showed large differences between these two algorithms for these instances, in favor of ALG_6 for MyBatis, and in favor of HITO for BCEL and JHotDraw.

Nevertheless, ALG_6 obtained large or medium differences when compared to SPEA2 and NSGA-II for the 5 biggest problems. The same does not occur to HITO, since it lost to SPEA2 with large difference in the AJHSQLDB instance and to NSGA-II in the AJHSQLDB instance with small difference. This emphasizes even more that GEMOITO can overcome the results of both conventional MOEAs. When comparing to HITO, ALG_6 is able to obtain large and favorable differences for the 3 biggest instances, while obtaining worse results for the next 2 biggest ones. Even though this showed that HITO is better in some instances, these results do not get far from what we observed using the Kruskal-Wallis test: ALG_6 is, overall, a more robust algorithm.

An interesting point we noted is that HITO was able to outperform all other algorithms (with statistical difference sometimes) in the BCEL instance. Even though the PF_{known} fronts are a bit mixed for this instance, it has a different search space when compared to the other ones, and obtaining the best solutions in this case may require different configurations. This was noted by Guizzo et al. [15], since, only for this instance, the most selected operators were different from the most selected operators of the other instances. What happens is that HITO can dynamically select the best operators for different instances. GEMOITO does not have this ability, since it was trained in a rather conventional instance (AJHSQLDB) of the problem, thus it could not adapt so well as HITO did. Ultimately, however, GEMOITO can generate powerful algorithms that can outperform such dynamism of an online hyper-heuristic.

The major drawback of GEMOITO is that it can generate very powerful algorithms that can perform similarly to HITO (ALG_1, ALG_2, ALG_5 and ALG_7), and even algorithms (ALG_6) that can outperform both conventional MOEAs and also HITO. However, GEMOITO can also generate algorithms that perform badly (ALG_7 and ALG_9) when compared to the others. In a real world situation, the engineer might execute GEMOITO only once, which can result in a very powerful algorithm or a not so good one. A solution for this would be to improve the training process in order to consider multiple instances, which can prevent over-fitting to one instance of the problem and then turning the obtained algorithm more generic. In spite of that, GEMOITO seems a more plausible approach than using conventional algorithms, since even the worst generated algorithms obtained PF_{known} fronts and hypervolume values very close or even better than the conventional MOEAs.

4.3 Answering the Research Questions

We can positively answer the first question of this empirical evaluation: GEMOITO can generate MOEAs that are better than conventional ones. This is true for the problem instances in which we executed GEMOITO and using the hypervolume quality indicator with both statistical tests. If other indicators or statistical tests were used, then other results could be achieved. However, we still expect to obtain

Table 4: Hypervolume averages

Problem	NSGA-II	SPEA2	HITO	ALG_0	ALG_1	ALG_2	ALG_3	ALG_4	ALG_5	ALG_6	ALG_7	ALG_8	ALG_9
MyBatis	0.65	0.62	**0.71**	0.69	0.66	0.65	0.62	0.63	**0.73**	**0.74**	0.48	**0.76**	0.69
AJHsqldb	0.34	0.37	0.31	0.54	**0.65**	**0.64**	**0.66**	**0.65**	**0.65**	**0.70**	0.53	**0.60**	0.57
AJHotDraw	0.24	0.48	0.69	0.68	**0.87**	**0.86**	**0.80**	**0.84**	0.64	**0.87**	0.52	0.58	0.50
BCEL	0.73	0.70	**0.78**	0.74	0.73	0.68	0.72	0.69	**0.75**	**0.76**	0.47	**0.75**	0.64
JHotDraw	0.43	0.60	**0.85**	**0.67**	**0.77**	**0.74**	0.67	0.67	0.57	**0.69**	0.50	0.60	0.55
HealthWatcher	0.89	**0.98**	**0.99**	**0.94**	**1.0**	**1.0**	**1.0**	**1.0**	**0.99**	**0.99**	**0.97**	**0.99**	**0.98**
JBoss	**0.88**	**0.95**	**1.0**	**0.92**	**1.0**	**1.0**	**0.98**	**1.0**	**0.89**	**0.94**	**0.95**	**0.94**	**0.89**

Table 5: Effect Size results

System	NSGA-II/SPEA2	NSGA-II/HITO	NSGA-II/ALG_6	SPEA2/HITO	SPEA2/ALG_6	HITO/ALG_6
MyBatis	0.49 (small)	-1.22 (large)	-1.81 (large)	-2.08 (large)	-2.66 (large)	-0.97 (large)
AJHsqldb	-0.30 (small)	0.30 (small)	-3.54 (large)	0.89 (large)	-4.62 (large)	-5.69 (large)
AJHotDraw	-1.92 (large)	-3.86 (large)	-6.63 (large)	-1.65 (large)	-3.64 (large)	-1.87 (large)
BCEL	0.62 (medium)	-1.34 (large)	-0.96 (large)	-6.06 (large)	-4.63 (large)	2.82 (large)
JHotDraw	-0.83 (large)	-2.21 (large)	-1.30 (large)	-1.47 (large)	-0.51 (medium)	0.90 (large)
HealthWatcher	-1.07 (large)	-1.31 (large)	-1.32 (large)	-0.42 (small)	-0.44 (small)	-0.040 (negligible)
JBoss	-0.36 (small)	-0.64 (medium)	-0.30 (small)	-0.51 (medium)	0.077 (negligible)	0.53 (medium)

better results given the robustness of the generated algorithms. We intend to investigate this in future works.

Regarding the second question and taking into account only the quality of the results, then yes, GEMOITO can generate MOEAs that can generally outperform an online hyper-heuristic (such as HITO). However, there are other factors that must be taken into account. HITO is more dynamic and thus can adapt to different instances of the problem, which enables it to obtain more balanced results, but sometimes worse than the results of the generated MOEAs. On the other hand, GEMOITO does not require the selection of a MOEA, which can reduce the testers' effort. Hence, the choice depends on the preferences of the engineer, and all of these differences should be evaluated when deciding which hyper-heuristic will be used.

5. CONCLUDING REMARKS

This paper presented an offline hyper-heuristic for generating MOEAs to solve the ITO problem. GEMOITO is based on GE, and includes a grammar with several components and parameters related to the MOEA design. During the evolution, GEMOITO executes the generated MOEAs and the best ones survive according to the hypervolume quality indicator. At the end, the best trained MOEA is returned and can be used to solve the problem.

For assessing the GEMOITO applicability, we conducted an empirical evaluation in two phases: i) training; and ii) testing. In the training phase we executed GEMOITO 10 times, which resulted in 10 MOEAs. After that, in the testing phase we compared the generated algorithms with two conventional MOEAs (NSGA-II and SPEA2) and with another hyper-heuristic applied to the same problem (HITO) using 7 real world systems. The empirical evaluation showed that, even though not all generated algorithms can outperform HITO, all of them obtained good results when compared to the conventional MOEAs. Furthermore, some generated MOEAs at least equaled HITO and one of them was able to obtain the best results in all systems. We positively answered the research questions stating that GEMOITO is viable and can generate robust algorithms.

As future works we intend to change some aspects of the training procedure used in GEMOITO, mainly to balance the behavior of the generated MOEAs. Another possibility is to use other quality indicators as fitness functions for this training. We want to perform other experiments, with a greater number of large systems, other MOEAs and other hyper-heuristics (such as [3, 4]). Finally, some changes could be made in the proposed grammar to allow more intelligent choices of numeric parameters, such as a percentage of the population as tournament size or a search space-based function for population size.

References

[1] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, and A. Pozo. A multi-objective optimization approach for the integration and test order problem. *Information Sciences*, 267(0):119 – 139, 2014.

[2] M. P. Basgalupp, R. C. Barros, T. S. da Silva, and A. C. Carvalho. Software Effort Prediction: A Hyper-heuristic Decision-tree Based Approach. In *Symposium on Applied Computing*, pages 1109–1116, 2013.

[3] L. Bezerra, M. Lopez-Ibanez, and T. Stuetzle. Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, PP(99):1–1, 2015.

[4] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic Design of Evolutionary Algorithms for Multi-Objective Combinatorial Optimization. In *Parallel Problem Solving from Nature - PPSN*, volume 8672, pages 508–517. Springer, 2014.

[5] L. C. Briand, J. Feng, and Y. Labiche. Using genetic algorithms and coupling measures to devise optimal integration test orders. In *Conference on Software Engineering and Knowledge Engineering*, July 2002.

[6] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[7] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A Classification of Hyper-heuristic Approaches. In *Handbook of Metaheuristics*, v. 146, pages 449–468. Springer, 2010.

[8] E. K. Burke, M. R. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3):406 – 417, 2012.

[9] J. Cohen. A power primer. *Psychological bulletin*, 112(1):155, 1992.

[10] 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, 2002.

[11] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 3–18, 2011.

[12] J. Dréo. Using performance fronts for parameter setting of stochastic metaheuristics. In *Conference on Genetic and Evolutionary Computation: Late Breaking Papers (GECCO'09)*, pages 2197–2200, 2009.

[13] A. Eiben and S. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, March 2011.

[14] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

[15] G. Guizzo, G. M. Fritsche, S. R. Vergilio, and A. T. R. Pozo. A Hyper-Heuristic for the Multi-Objective Integration and Test Order Problem. In *Genetic and Evolutionary Computation Conference*, 2015.

[16] M. Harman, E. Burke, J. Clarke, and X. Yao. Dynamic adaptive search based software engineering. In *Proceedings of the 6th ESEM*, 2012.

[17] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1), 2012.

[18] Y. Jia. Hyperheuristic search for SBST. In *International Workshop on Search-Based Software Testing*, pages 15–16, 2015.

[19] Y. Jia, M. Cohen, M. Harman, and J. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. In *International Conference on Software Engineering (ICSE'15)*, 2015.

[20] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[21] A. C. Kumari, K. Srinivas, and M. P. Gupta. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In *Proceedings of the 3rd IACC*, pages 813–818, Feb. 2013.

[22] K. Li, A. Fialho, S. Kwong, and Q. Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, Feb 2014.

[23] M. Lopez-Ibanez and T. Stutzle. The Automatic Design of Multiobjective Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[24] N. Lourenço, F. B. Pereira, and E. Costa. The optimization ability of evolved strategies. In *Progress in Artificial Intelligence*, v. 9273, pages 226–237. 2015.

[25] N. Lourenço, F. Pereira, and E. Costa. Evolving evolutionary algorithms. In *Companion of the Genetic and Evolutionary Computation Conference (GECCO'12)*, page 51, 2012.

[26] N. Lourenço, F. B. Pereira, and E. Costa. The Importance of the Learning Conditions in Hyper-heuristics. In *Companion of the Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 1525–1532, 2013.

[27] M. Maashi, E. Özcan, and G. Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications*, 41(9):4475–4493, 2014.

[28] R. Marshall, M. Johnston, and M. Zhang. Developing a hyper-heuristic using grammatical evolution and the capacitated vehicle routing problem. In *Simulated Evolution and Learning*, volume 8886, pages 668–679. Springer, 2014.

[29] R. J. Marshall, M. Johnston, and M. Zhang. Hyper-heuristics, grammatical evolution and the capacitated vehicle routing problem. In *Companion of the Genetic and Evolutionary Computation Conference (GECCO'14)*, pages 71–72, 2014.

[30] F. Mascia, M. Lopez-Ibanez, J. Dubois-Lacoste, and T. Stutzle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*, 51:190 – 199, 2014.

[31] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[32] C. Ryan, J. J. Collins, and M. Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, volume 1391, pages 83–96. Springer, 1998.

[33] S. K. Smit, A. E. Eiben, and Z. Szlávik. An MOEA-based method to tune EA parameters on multiple objective functions. In *Proceedings of the 2nd International Joint Conference on Computational Intelligence (IJCCI'10)*, 2010.

[34] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: improving the strength Pareto evolutionary algorithm. Technical report, Dep. of Electrical Engineering, Swiss Federal Institute of Technology, 2001.

[35] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.