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

Conference Paper · July 2016
DOI: 10.1145/2908812.2908816

4 authors:

Thainá Mariani
Universidade Federal do Paraná
7 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Giovani Guizzo
Universidade Federal do Paraná
17 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Silvia Regina Vergilio
Universidade Federal do Paraná
133 PUBLICATIONS **899** CITATIONS
SEE PROFILE

Aurora Pozo
Universidade Federal do Paraná
142 PUBLICATIONS **809** CITATIONS
SEE PROFILE

A Grammatical Evolution Hyper-Heuristic for the 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 software testing problems. Still, the decision of choosing a search technique and configuring its parameters can be a hard task. Hyper-heuristics are promising solutions for freeing the tester from this decision. In this context, this paper presents an offline hyper-heuristic based on Grammatical Evolution (GE) to generate a Multi-Objective Evolutionary Algorithms (MOEA) to solve the Integration and Test Order (ITO) problem. The MOEAs are distinguished by different components and parameters values. The MOEAs generated by the proposed hyper-heuristic are evaluated using seven instances of the ITO problem. They are compared to conventional MOEAs and with a hyper-heuristic used in related work. Results show that the proposed hyper-heuristic can generate MOEAs that are statistically equivalent or better 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

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

1. INTRODUCTION

Search Based Software Engineering (SBSE) addresses search techniques to solve software engineering problems. SBSE is widely used and was successfully applied in many software engineering areas, such as requirements, design, maintenance and testing [16]. SBSE is a very attractive option to solve hard software engineering problems, but by using search techniques, many decisions have to be made by the engineer [12]. Firstly, it is necessary to choose the search technique 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, and will most likely affect the final result. For instance, in a simple genetic algorithm, the engineer needs to configure the population size, crossover operator, crossover probability, mutation operator and mutation probability. Therefore, deciding all these aspects is a hard task that can be considered an optimization problem by itself [12]. In this sense, hyper-heuristics can help the engineer by selecting or generating heuristics to solve a problem [6].

Hyper-heuristics [6] are heuristics used for selecting or generating heuristics to solve hard computational search problems. A learning mechanism can be used by the hyper-heuristic for updating the preference of each heuristic based on its historical performance. It can be performed while the instance of a problem is being solved, representing an online learning. Furthermore, the learning can be first performed in a set of training instances and then, the resulted heuristic is used to solve other instances, representing an offline learning. Hence, hyper-heuristics can be classified according to the nature of the heuristic search space (selection or generation) and the source of feedback during learning (online, offline or no-learning) [6].

Guizzo et al. [14] proposed HITO (A Hyper-heuristic for the Integration and Test Order Problem), an online selection hyper-heuristic applied for the Integration and Test Order (ITO) problem. HITO selects the best low-level heuristic, consisting of a combination of a crossover operator and a mutation operator used by Evolutionary Algorithms (EAs). At each mating, the best low-level heuristic is applied. In order to do this, two selection functions are available: Choice Function (CF) [25] and Multi-Armed Bandit (MAB) [20].

ITO is a hard software engineering problem explored by many works, using different algorithms [31]. The problem consists to determine a sequence to integrate and test software units, usually associated to minimal cost. Most works associate the sequence cost to many factors, such as number of attributes and number of methods. In this context, search techniques are promising options to solve this problem, since they can be used to perform multi-objective optimization [31]. Guizzo et al. [14] point out that the application of hyper-heuristic in this problem is very suitable by being properly solved by multi-objective algorithms. Furthermore, due to the permutation representation of the problem, there are many operators to be selected [14]. Nevertheless, until now, HITO was the first work to explore the ITO problem using hyper-heuristic. HITO obtained encouraging results. Moreover, the engineer does not have to worry about

GECCO '16, July 20 - 24, 2016, Denver - Colorado, USA
© 2016 ACM. ISBN XXXXXXXXXXXXX…$XX.XX
DOI: XXXXXXXXXXXX

choosing a specific operator, but he/she still has to choose a specific algorithm and to select the other components and parameters values of the algorithm.

In this sense, we propose an offline hyper-heuristic to generate Multi-Objective Evolutionary Algorithms (MOEAs) for solving the ITO problem, named GHITO (Grammatical Evolution Hyper-Heuristic for the Integration and Test Order Problem). The generated MOEAs are distinguished by many components and parameters, including population initialization, selection, mating, operators, replacement and archiving. We chose the generation of MOEAs because they are multi-objective algorithms and because existing MOEAs were already explored for this problem [1, 14]. That way, we believe that a new MOEA, trained specifically in 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 explored the generation of EAs in other contexts [3, 4, 23], presenting promising results.

GHITO is based on Grammatical Evolution (GE) [29]. GE is a type of Genetic Programming (GP) [18] that uses a grammar to generate programs. The grammar defines rules to be used in the GE evolutionary process in order to find better programs. Several works in the literature use GE to generate search algorithms [7, 22–24, 26–28], some of them are related to the generation of EAs [22–24]. Results obtained by these works encouraged us to use GE as the heuristic for generating MOEAs. The GE grammar is composed by several possible values for the components and parameters of MOEAs, therefore the GE algorithm can generate a MOEA containing the best combination of components and parameters values.

An empirical evaluation was conducted using 7 instances of the ITO problem. Experiments were performed comparing the results obtained by GHITO, HITO and the conventional MOEAs: Nondominated Sorting Genetic Algorithm II (NSGA-II) [9] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [32]. The obtained Pareto fronts were evaluated using the hypervolume indicator [33], the Kruskal-Wallis statistical test [10] and the Effect Size statistical test [8]. The experiments showed that GHITO results are statistically equivalent or better than HITO, NSGA-II and SPEA2.

This paper is organized as follows. Section 2 summarizes related work. Section 3 presents GHITO, containing the grammar, defined components and parameters values. Section 4 presents the empirical evaluation, containing the research questions, details about the instances, the performed experiments and the obtained results. Finally, Section 5 concludes this work and discusses some future works.

2. RELATED WORK

Hyper-heuristics were explored by many works in the literature [5]. In the SBSE context, hyper-heuristics can contribute to obtain a holistic and generic SBSE [15]. However, just few works are found in the literature encompassing hyper-heuristics and SBSE, which 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. [19] 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. [17] investigate a simulated annealing hyper-heuristic to apply strategies of the Combinatorial Interaction Testing (CIT) problem. The hyper-heuristic performs an online learning to apply dynamically the best CIT strategy. Jia [17] presented an idea of a selection hyper-heuristic framework for Search Based Software Testing (SBST) problems. 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, but the author presented an example using the CIT problem.

Guizzo et al. [14] proposed HITO, an online selection hyper-heuristic for the ITO problem. 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. The hyper-heuristic implements two selection functions to select the low-level heuristics: Choice Function (CF) [25] and Multi-Armed Bandit (MAB) [20]. 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. HITO results outperformed the ones obtained by some conventional MOEAs. These results encouraged us to propose other hyper-heuristic for the ITO problem. Furthermore, we believe that changing other MOEAs components, in addition to changing only crossover and mutation operators, can improve the results. For this purpose, we explored the generation of MOEAs, which are distinguished by components and parameters values.

There are some works in the literature [7, 22–24, 26–28] using GE for the generation of search algorithms. Lourenço et al. [22–24] focus on the automatic generation and tuning of mono-objective EAs. However, we did not find 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, 11, 21, 30]. In the MOEA context, most works are related to the parameter tuning [11, 30]. 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. In the software engineering context, we did not find works addressing hyper-heuristics for the generation of MOEAs.

The choice of using GE to generate MOEAs was mainly inspired by [22–24], 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. Based on their works, we modeled these components and parameters in the GHITO 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.

3. GHITO

Grammatical Evolution Hyper-Heuristic for the Integration and Test Order Problem (GHITO) is an offline generation hyper-heuristic based on GE used 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) [6]. One of the advantages of such hyper-heuristics is that, even though they usually need a great amount of resources to train the low-level heuristics, no resources are wasted in the training during the problem solving. Furthermore, the trained heuristics are expected to be reusable throughout the problem instances. Generation hyper-heuristics are used to generate low-level heuristics, instead of selecting existing ones as selection hyper-heuristics do [6]. The generated heuristics are expected to perform well on unknown instances of the problem, thus usually some sort of training (mainly offline) is employed to find such robust heuristics.

GE [29] algorithms are a type of Genetic Programming (GP) [18] 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 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 GE algorithm keeps consuming genes and using them to choose rule options until a full solution (tree) is constructed. At the end, the assembled 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 what are the possible nodes for the solution.

The objective of GHITO is to generate MOEAs, its grammar contains rules that define the components, parameters and their values available for generating and configuring MOEAs. Such 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 represents a `null` option.

The rules of GHITO's 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, $\langle populationSize \rangle$ denotes the population size parameter and everything after "::=" are values that can be used for this parameter. Similarly, $\langle crossoverOperator \rangle$ 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" [13]. Note that, because ITO is a permutation problem, all the crossover and mutation operators available in GHITO's grammar are for permutation solutions. The first rule of the grammar ($\langle GA \rangle$) defines what are the rules that can be used by the GE algorithm to generate a MOEA. The grammar defines which are the available components, parameters and their respective values, but the template of the MOEAs is the same as shown in Algorithm 1.

```
⟨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: GHITO's grammar

What the GE mapper actually does is to select the rule options according to the genotype and 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. GHITO receives this MOEA and executes it using a training instance of the problem. The resulting Pareto front is then evaluated using hypervolume [33] 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

Algorithm 1: Template of the generated MOEAs

```
1  begin
2      $population$ ← Population initialization;
3      Evaluate ($population$);
4      Archive ($population$);
5      while $stop\ creteria\ is\ not\ achieved$ do
6          $matingPopulation$ ← Selection ($population$);
7          $offspringPopulation$ ← Crossover
           ($matingPopulation$);
8          $offspringPopulation$ ← Mutation
           ($offspringPopulation$);
9          Evaluate ($offspringPopulation$);
10         Replacement ($offspringPopulation$, $population$);
11         Archive ($offspringPopulation$);
```

the grammar, but at the end only the best MOEA is given as result to solve the ITO problem.

The ITO problem is 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 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 the unit B first and then the unit A (sequence of {B, A}), thus no stubs are required. However, in large systems it is not that simple since there are lots of units and potentially several cyclic associations (e.g. A requires B and B requires A; or A requires B, B requires C and C requires A) that cannot be broken without a stub. The idea is to reduce the testing cost by minimizing the resources employed into developing stubs.

ITO uses a permutation representation for its solutions, where each gene is the ID of a unit. 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]. The generated MOEAs use two objective functions in this work: number of operations (O) and number of attributes (A). These two functions can be used in object-oriented and aspect-oriented software to measure the number of required methods/operations and number of attributes to be emulated.

One important thing to note is that when dealing with multiple objectives, there might be incomparable solutions, as known as non-dominated solutions. 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 [9] uses Dominance Depth and Crowding Distance, whereas SPEA2 [32] uses Raw Fitness and K-th Nearest Neighbor. GHITO'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, GHITO lets the generated MOEAs evaluate each solution in three different ways (one for each procedure) focusing more on convergence or diversity depending the selected strategy. We expect to obtain more robust MOEAs using such approach.

4. EMPIRICAL EVALUATION

In order to evaluate GHITO, we conducted an empirical evaluation using HITO, conventional MOEAs and the ones generated by GHITO in 7 real world systems. The empirical evaluation is divided in two phases: i) training; and ii) testing. In the training phase, we executed GHITO 10 times in order to generate 10 MOEAs. Also in this phase we used GHITO to automatically tune the conventional MOEAs in order to obtain a fair comparison between all MOEAs. 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. The results were evaluated using the obtained Pareto fronts, the hypervolume indicator [33], the Kruskal-Wallis statistical test [10] and the Cohen's d effect size [8].

Subsection 4.1 presents the research questions for this study. Subsection 4.2 shows the systems used for the training and testing phases. Subsections 4.3 and 4.4 describe respectively the training and testing phases of this evaluation. Finally, Subsection 4.5 answers the research questions.

4.1 Research Questions

This experimentation is guided by two main Research Questions (RQs):

1. Can GHITO generate MOEAs that can outperform conventional MOEAs used in the literature?

2. Can GHITO generate MOEAs that can outperform another hyper-heuristic used for solving the same problem?

For answering the first question, we compared the MOEAs generated by GHITO with two conventional MOEAs: Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [9] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [32]. For the second question, we used HITO as hyper-heuristic for solving the ITO problem, since it was already tested in the same systems and it is a recent and powerful hyper-heuristic. Another motivation for choosing HITO is that it is an online selection hyper-heuristic, which contrasts with GHITO (an offline generation hyper-heuristic). By comparing them both, we can observe the benefits and disadvantages of both kinds of hyper-heuristics.

4.2 Systems

We used 7 real world systems for this evaluation, which are presented in Table 1. These are the same systems used by HITO in [14] and by another related work using conventional MOEAs [1]. They vary in the implemented paradigm (Object Oriented – OO, or Aspect Oriented – AO), number of units, number of dependencies and number of Lines of Code (LOC).

4.3 Training

We fixed the parameters of GHITO after a preliminary experimentation using similar values to the work of Lourenco et al. [23]. Table 2 presents these parameters.

Table 1: Systems used in the experiments

Name	Paradigm	Units		Dep.	LOC
		Class	Aspect		
MyBatis	OO	331	–	1271	23535
AJHSQLDB	AO	276	25	1338	68550
AJHotDraw	AO	290	31	1592	18586
BCEL	OO	45	–	289	2999
JHotDraw	OO	197	–	809	20273
HealthWatcher	AO	95	22	399	5479
JBoss	OO	150	–	367	8434

Table 2: GHITO 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

With these parameters, GHITO was executed 10 times using the training instance AJHSQLDB, resulting in 10 different MOEAs (here named ALG_0 to ALG_9). Each MOEA was trained with a budget of 2,000 fitness evaluations. The best algorithm found (as we show in the testing subsection 4.4) is ALG_6. Its components and parameters are presented in Table 3. 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.

Table 3: 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

Similarly, we used GHITO with slightly different grammars for automatically tuning both NSGA-II and SPEA2. Due to space restrictions, this tunning is omitted. The HITO parameter configuration is the same as presented in [14], 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 [14]. The parameters of HITO, NSGA-II and SPEA2 are presented in Table 4. HITO does not use crossover and mutation probabilities because it dynamically applies the operators according to the search stage.

Table 4: 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	-	-

4.4 Testing

In the testing phase, first we executed all 13 algorithms (NSGA-II, SPEA2, HITO and the 10 MOEAs generated by GHITO) in the 7 real world systems for 30 independent runs using 60.000 fitness evaluations each run. As 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}). Because these are real systems, there are no true Pareto fronts for the problem.

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

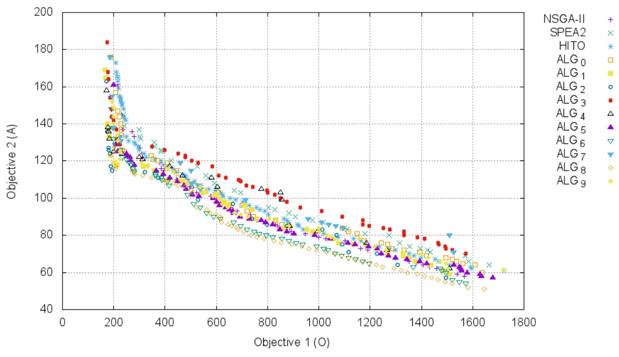

Figure 2: PF_{knwon} fronts found for MyBatis

The fronts found by NSGA-II, SPEA2 and HITO for AJHSQLDB (Figure 3) are dominated by at least one algorithm generated by GHITO, 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 happened for other instances, such as MyBatis (Figure 2) and AJHotDraw (Figure 4). We observed that the fronts of HITO and the generated algorithms (mostly ALG_6) remained relatively close when compared to

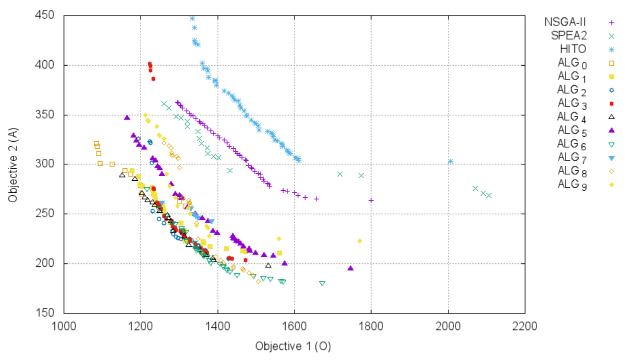

Figure 3: PF_{knwon} fronts found for AJHSQLDB

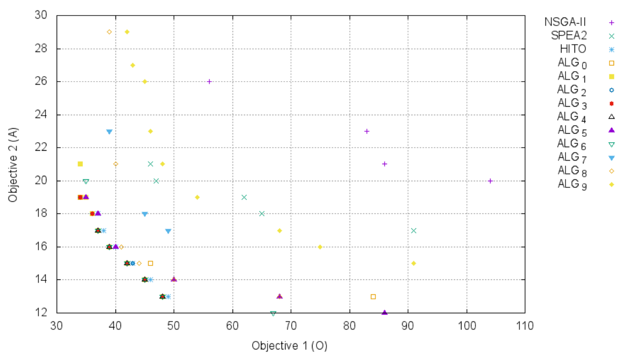

Figure 4: PF_{knwon} fronts found for AJHotDraw

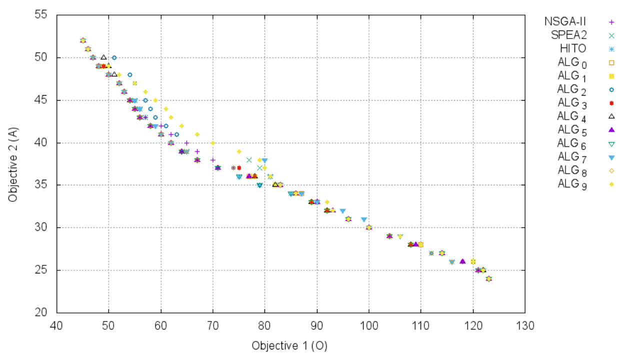

Figure 5: PF_{knwon} fronts found for BCEL

the fronts of NSGA-II and SPEA2. Thus, we can state that both hyper-heuristics found the best results.

Since most of the fronts are mixed in the objective space, we also used the hypervolume indicator [33] to calculate the quality of the fronts of each algorithm in order to better evaluate the results. This indicator was chosen due to its capability of evaluating both the convergence and diversity of the fronts, while also not needing a true Pareto front for its assessment. Table 5 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 values equivalent to the best ones according to the Kruskal-Wallis statistical test with 95% confidence.

As seen in Table 5, the conventional MOEAs NSGA-II and

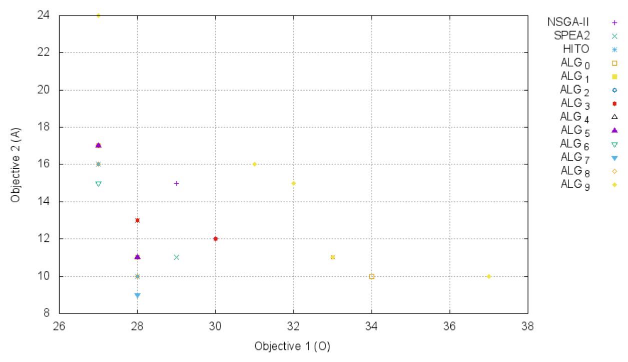

Figure 6: PF_{knwon} fronts found for JHotDraw

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 had the best results. Furthermore, for all systems, at least one algorithm generated by GHITO obtained greater hypervolume averages than both NSGA-II and SPEA2. The HITO results were similar as obtained in [14] regarding the comparison with the conventional MOEAs.

Comparing HITO and GHITO, for 2 out of the 7 systems, 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 of the 10 algorithms generated by GHITO 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 [8] effect size. This statistical test gives the difference magnitude between two groups of values. In our work, the groups are the algorithms and their values are the 30 hypervolume values calculated using the Pareto fronts obtained with the 30 independent runs. Table 6 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 result showed 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 GHITO can overcome the results of both conventional MOEAs. When com-

Table 5: 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	6,5E-1	6,2E-1	**7,1E-1**	6,9E-1	6,6E-1	6,5E-1	6,2E-1	6,3E-1	**7,3E-1**	**7,4E-1**	4,8E-1	**7,6E-1**	6,9E-1
AJHsqldb	3,4E-1	3,7E-1	3,1E-1	5,4E-1	**6,5E-1**	**6,4E-1**	**6,6E-1**	**6,5E-1**	**6,5E-1**	**7,0E-1**	5,3E-1	**6,0E-1**	5,7E-1
AJHotDraw	2,4E-1	4,8E-1	6,9E-1	6,8E-1	**8,7E-1**	**8,6E-1**	**8,0E-1**	**8,4E-1**	6,4E-1	**8,7E-1**	5,2E-1	5,8E-1	5,0E-1
BCEL	7,3E-1	7,0E-1	**7,8E-1**	7,4E-1	7,3E-1	6,8E-1	7,2E-1	6,9E-1	**7,5E-1**	**7,6E-1**	4,7E-1	**7,5E-1**	6,4E-1
JHotDraw	4,3E-1	6,0E-1	**8,5E-1**	**6,7E-1**	**7,7E-1**	**7,4E-1**	6,7E-1	6,7E-1	5,7E-1	**6,9E-1**	5,0E-1	6,0E-1	5,5E-1
HealthWatcher	8,9E-1	**9,8E-1**	**9,9E-1**	**9,4E-1**	**1,0E0**	**1,0E0**	**1,0E0**	**1,0E0**	**9,9E-1**	**9,9E-1**	**9,7E-1**	**9,9E-1**	**9,8E-1**
JBoss	**8,8E-1**	**9,5E-1**	**1,0E0**	**9,2E-1**	**1,0E0**	**1,0E0**	**9,8E-1**	**1,0E0**	**8,9E-1**	**9,4E-1**	**9,5E-1**	**9,4E-1**	**8,9E-1**

Table 6: Effect Size results

System	NSGA-II/SPEA2	NSGA-II/HITO	NSGA-II/ALG_6	SPEA2/HITO	SPEA2/ALG_6	HITO/ALG_6
MyBatis	4,90E-1 (small)	-1,22E0 (large)	-1,81E0 (large)	-2,08E0 (large)	-2,66E0 (large)	-9,71E-1 (large)
AJHsqldb	-3,08E-1 (small)	3,01E-1 (small)	-3,54E0 (large)	8,95E-1 (large)	-4,62E0 (large)	-5,69E0 (large)
AJHotDraw	-1,92E0 (large)	-3,86E0 (large)	-6,63E0 (large)	-1,65E0 (large)	-3,64E0 (large)	-1,87E0 (large)
BCEL	6,27E-1 (medium)	-1,34E0 (large)	-9,64E-1 (large)	-6,06E0 (large)	-4,63E0 (large)	2,82E0 (large)
JHotDraw	-8,39E-1 (large)	-2,21E0 (large)	-1,30E0 (large)	-1,47E0 (large)	-5,12E-1 (medium)	9,08E-1 (large)
HealthWatcher	-1,07E0 (large)	-1,31E0 (large)	-1,32E0 (large)	-4,25E-1 (small)	-4,45E-1 (small)	-4,03E-2 (negligible)
JBoss	-3,64E-1 (small)	-6,48E-1 (medium)	-3,00E-1 (small)	-5,10E-1 (medium)	7,79E-2 (negligible)	5,31E-1 (medium)

paring 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 instances. 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. [14], 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 such as this. GHITO 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, GHITO can generate powerful algorithms that can outperform such dynamism of an online hyper-heuristic.

The major drawback of GHITO 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, GHITO 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 GHITO 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 balance the algorithms and give them a balanced behavior. In spite of that, GHITO 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.

4.5 Answering the Research Questions

We can positively answer the first question of this empirical evaluation: GHITO can generate MOEAs that are better than conventional ones. This is true for the problem instances in which we executed GHITO 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 better results given the robustness of the generated algorithms. We intend to investigate this in future works.

Regarding the second question, if take only into account the quality of the results, then yes, GHITO 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. It all depends on what the engineer really wants, and all of these differences must be evaluated when deciding which hyper-heuristic he/she will be using.

5. CONCLUDING REMARKS

This paper presented an offline hyper-heuristic for generating MOEAs used to solve the ITO problem. GHITO is based on grammatical evolution, which uses a grammar with several components and parameters to generate MOEAs. During the evolution, GHITO 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 applicability of GHITO, we conducted an empirical evaluation consisting in two phases: i) training; and ii) testing. In the training phase we executed GHITO 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 of the 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 GHITO is viable and can generate robust algorithms.

As future works we intend to change some aspects of the training procedure used in GHITO, mainly to balance the behavior of the generated MOEAs. Another possibility is to use other quality indicators as fitness functions for this training. We also want to perform other experiments, with a greater number of large systems, with more conventional MOEAs and with other hyper-heuristics.

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. P. L. F. Carvalho. Software Effort Prediction: A Hyper-heuristic Decision-tree Based Approach. In *Proceedings of the 28th SAC*, pages 1109–1116. ACM, 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 XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 508–517. Springer International Publishing, 2014.

[5] 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.

[6] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A Classification of Hyper-heuristic Approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010.

[7] 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.

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

[9] 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.

[10] 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*, 1(1):3–18, 2011.

[11] J. Dréo. Using performance fronts for parameter setting of stochastic metaheuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers (GECCO'09)*, pages 2197–2200, New York, NY, USA, 2009. ACM.

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

[13] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer Science & Business Media, 2003.

[14] 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 *Proceedings of the 24th Genetic and Evolutionary Computation Conference (GECCO'15)*. ACM, 2015.

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

[16] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, 2012.

[17] Y. Jia, M. Cohen, M. Harman, and J. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, May 2015. To appear.

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

[19] 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.

[20] 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.

[21] 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.

[22] N. Lourenço, F. B. Pereira, and E. Costa. The optimization ability of evolved strategies. In F. Pereira, P. Machado, E. Costa, and A. Cardoso, editors, *Progress in Artificial Intelligence*, volume 9273 of *Lecture Notes in Computer Science*, pages 226–237. Springer International Publishing, 2015.

[23] N. Lourenço, F. Pereira, and E. Costa. Evolving evolutionary algorithms. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion - GECCO Companion '12*, page 51, New York, New York, USA, jul 2012. ACM Press.

[24] N. Lourenço, F. B. Pereira, and E. Costa. The Importance of the Learning Conditions in Hyper-heuristics. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO'13)*, pages 1525–1532, New York, NY, USA, 2013. ACM.

[25] 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.

[26] R. Marshall, M. Johnston, and M. Zhang. Developing a hyper-heuristic using grammatical evolution and the capacitated vehicle routing problem. In G. Dick, W. Browne, P. Whigham, M. Zhang, L. Bui, H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. Tan, and K. Tang, editors, *Simulated Evolution and Learning*, volume 8886 of *Lecture Notes in Computer Science*, pages 668–679. Springer International Publishing, 2014.

[27] R. J. Marshall, M. Johnston, and M. Zhang. Hyper-heuristics, grammatical evolution and the capacitated vehicle routing problem. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 71–72, New York, NY, USA, 2014. ACM.

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

[29] C. Ryan, J. J. Collins, and M. Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–96. Springer Berlin Heidelberg, 1998.

[30] 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.

[31] Z. Wang, B. Li, L. Wang, and Q. Li. A brief survey on automatic integration test order generation. In *Proceedings of the 23rd SEKE*, pages 254–257, 2011.

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

[33] 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.