A Hyper-heuristic for Multi-Objective Integration and Test Ordering in Google Guava

Giovani Guizzo[1], Mosab Bazargani[2], Matheus Paixao[3] and John H. Drake[2]

[1] Federal University of Paraná (UFPR)
gguizzo@inf.ufpr.br
[2] Operational Research Group, Queen Mary University of London,
Mile End Road, London, E1 4NS, UK
{m.bazargani, j.drake}@qmul.ac.uk
[3] CREST, Department of Computer Science, University College London,
Gower Street, London, WC1E 6BT, UK
matheus.paixao.14@ucl.ac.uk

Abstract. Integration testing seeks to find communication problems between different units of a software system. As the order in which units are considered can impact the overall effort required to perform integration testing, deciding an appropriate sequence to integrate and test units is vital. Here we apply a multi-objective hyper-heuristic set within an NSGA-II framework to the Integration and Test Order Problem (ITO) for Google Guava, a set of open-source common libraries for Java. Our results show that an NSGA-II based hyper-heuristic employing a simplified version of Choice Function heuristic selection, outperforms standard NSGA-II for this problem.

1 Introduction

The integration testing phase of a testing strategy combines and tests multiple units of a software system. As some units are dependent on others, stubs are used to mimic the behaviour of classes that are not available, are too expensive to use directly, or are not yet integrated and tested in the software. One drawback is that the stubbing process can also be expensive, and is potentially susceptible to errors. The Integration and Test Order Problem (ITO) is a search problem where the goal is to generate an order for units to be integrated and tested which minimises the cost of stub generation.

As there are a number of different ways of measuring stubbing cost, many previous approaches have considered ITO as a multi-objective problem. In multi-objective optimisation [3], where more than one objective is optimised at the same time, the aim is to find a set of solutions, known as the Pareto front, representing the best trade-off that exists between objectives. Assunção et al. [2] compared the performance of three well-known multi-objective evolutionary algorithms (MOEAs) for solving the ITO problem for eight software systems. Each of the MOEAs tested searched over a permutation of integers representing the order that units are integrated and tested, using two-point crossover and swap mutation to modify solutions. Separate performance comparison was provided for the ITO problem using two objectives and four objectives.

Hyper-heuristics are high-level search methods which operate over a search space of low-level heuristics or heuristic components, rather than over a search space of solutions directly. Guizzo et al. [5] built on the work of Assunção et al. [2], introducing HITO, a Hyper-heuristic for the Integration and Test Order Problem. Operating within the well-known multi-objective Non-dominated Sorting Genetic Algorithm II (NSGA-II) [3], HITO uses a heuristic selection method to select which operators to apply at each step of an MOEA from a set of crossover and mutation operator combinations, optimising two objectives. Using three different heuristic selection methods, their experiments showed that hyper-heuristic selection of crossover and mutation operators within NSGA-II outperformed the traditional NSGA-II implementation (in terms of hypervolume) using only 2-point crossover and swap mutation presented by Assunção et al. [2]. A further performance comparison of using HITO within an SPEA2 framework was given in a later paper [6]. Guizzo et al. [7] provided another extension, formulating the problem as a many-objective problem with four objectives, comparing to a number of state-of-the-art MOEAs.

Google Guava [1] is a large open-source project, containing Google versions of a number of standard general purpose libraries for Java. In this paper, we use all three versions of HITO presented by Guizzo et al. [7], to search for an optimal ordering of units for integration testing of Guava. A performance comparison to the original NSGA-II method presented by Assunção et al. [2] is given.

2 Problem description and solution methodology

The two first levels of tests in software testing are unit testing and integration testing. The unit testing level validates that each unit of the software performs as designed. Thereafter, integration testing is employed to expose faults in the interaction between integrated units. In this phase units are integrated into the software and then tested. When a unit is not yet integrated and tested, but its functionality is needed to integrate and test a dependent unit (an event which is known as dependency break), then a stub (emulation) must be created for such a unit. Generally speaking, units with a high number of calls (number of other units that are depending on them) should be integrated and tested prior to those units with a lower number of calls. If units are not tested in an optimised sequence, an extra cost for generating a greater number of stubs during integration testing will be imposed to the software testing process. Given n units to be integrated and tested, a solution to the problem is represented by a permutation of integers $[1, ..., n]$, denoting the order in which units are processed during integration testing.

This problem is a multi-objective optimisation problem, since several factors have an impact on the cost of stub construction, which makes it harder to find a good cost reduction. In this paper, we use the two objectives that are used by Guizzo et al. [5], i.e., number of attributes (A) and number of methods/operations (O); both need to be emulated in the stub if the dependencies between two modules are broken. Furthermore, this problem can be found in several development contexts. For example, in an object-oriented system units are classes, in component-based programming units are components, in aspect-oriented programming units are aspects, and in product line oriented systems units may be considered product features [2]. These characteristics make this problem suitable for the application of meta- and hyper-heuristics, since these kinds of algorithms are capable of optimizing several objectives at once [3], and are capable of being easily applied to different contexts without needing to have their implementation adapted for this end [7].

In order to extract the method/attribute dependencies of Google Guava we used the *Understand* tool, developed by scitools$^{\text{TM}}$ [8]. As *Understand* can only work with Java 1.7 or older versions, we extracted the dependences for Google Guava $v20.0$. We extracted two levels of dependencies, i.e., dependencies between units, and unit-method or unit-attribute dependencies. For each unit, *define*, *import*, *call*, and *override* features of that file are used for extracting its dependencies, and *define*, *use*, *set*, and *modify* features of each unit-method/unit-attribute are used for addressing dependencies of that method/attribute. Google Guava has 74530 lines of Java code, excluding comments and blank lines, written in 529 files. It has 2273 unit dependencies in total. This is bigger than all seven of the systems that HITO was applied to in previous work in the literature [5, 7], where the maximum number of unit dependencies was 1592.

3 HITO

Hyper-heuristic for the Integration and Test Order Problem (HITO) is an NSGA-II based hyper-heuristic for the ITO problem. Three different versions of HITO have been introduced in the literature [5], namely HITO-R, HITO-MAB, and HITO-CF. These versions operate within the same hyper-heuristic framework, using different heuristic selection methods. While HITO-R selects low-level heuristics randomly, HITO-MAB and HITO-CF try to provide balance between exploration and exploitation during the search. HITO-MAB uses a Multi-Armed Bandit (MAB) strategy, selecting low-level heuristics based on their performance and number of executions in a given number of iterations. HITO-CF employs a simplified variant of the Choice Function (CF) [4], using only the performance of a low-level heuristic (f_1) and the elapsed time since a low-level heuristic has been executed (f_3). The performance of pairs of low-level heuristics (f_2) is eliminated from the Choice Function used in HITO-CF for simplicity [5], as pairwise performance between different types of operators is difficult to assess within the NSGA-II framework.

HITO uses nine low-level heuristics, consisting of pairwise combinations of 2-point crossover, uniform crossover, and partially-mapped crossover (PMX), with swap mutation, simple insertion mutation or no mutation. All of the crossover and mutation operators used in HITO are permutation-based, since this is the representation used when considering ITO as a combinatorial optimisation problem. For more information on the heuristic selection methods and low-level heuristics used, we refer the interested reader to the original HITO paper [5].

4 Experiments

This section presents the set of experiments to evaluate the performance of the three different versions of HITO in the Google Guava program. We also compare those results with standard NSGA-II using 2-point crossover and swap mutation. The next two subsections present the experimental set-up and the results.

4.1 Experimental Set-up

The experimentation encompasses four algorithms for solving the ITO problem: HITO-MAB, HITO-CF, HITO-R and NSGA-II. All parameters were set as in Guizzo et al. [7]. All algorithms were executed for 30 independent runs on the unit dependencies extracted from Google Guava. For all of these algorithms, population size is set to 300 and stopping criterion to $60,000$ function evaluations. A crossover probability of 95% and mutation probability of 2% were used in NSGA-II experiments. HITO-MAB/CF/R dynamically select low-level heuristics as explained in the Section 3. A selected low-level heuristic is applied in HITO-MAB/CF/R with probability of 100%. The MAB parameters of HITO-MAB are size of the sliding window (W) and scaling factor (C) that are respectively set to 150 and 5. CF weight parameters of HITO-CF for f_1 and f_2 are set to $\alpha = 1.0$ and $\beta = 0.00005$, respectively.

The results were collected and evaluated using the hypervolume quality indicator [3]. Hypervolume is a measure of the volume of space dominated by the non-dominated set of solutions representing the approximation of the Pareto front, bounded by a given reference point.

4.2 Results

Table 1 shows hypervolume averages over 30 independent runs found from applying three different versions of HITO and NSGA-II to the Google Guava unit dependencies. Standard deviations of 30 independent runs are given in parenthesis. Hypervolumes are compared using the Kruskal-Wallis statistical test at 95% of significance, with the *p-value* reported alongside the hypervolume values in Table 1.

Table 1. Hypervolume averages obtained from 30 independent runs.

System	NSGA-II	HITO-CF	HITO-MAB	HITO-R	p-value
Google Guava	0.309 (0.126)	**0.685 (0.108)**	0.586 (0.085)	0.537 (0.083)	2.029E-15

As shown in Table 1, for Google Guava, HITO-CF performs statistically significantly better than other algorithms with a *p-value* of *2.029E-15*. All HITO versions, even HITO-R, performed better than NSGA-II on average, which indicates the need and effectiveness of using combinations of several low-level heuristics in this problem. This performance is broadly in line with the observations of Guizzo et al. in [7] for seven other systems, with HITO-CF outperforming HITO-R and NSGA-II, however in that work HITO-CF did not show statistically significantly different performance to HITO-MAB. As those seven systems have fewer lines of code and unit dependencies than Google Guava, it might be that HITO-CF scales better to larger systems.

To give a better understanding of the behaviour of the HITO variants, we examined the number of times that each low-level heuristic was executed by each hyper-heuristic. We observed that HITO-CF applied the low-level heuristics with 2-point crossover roughly 2.85 times more than those using uniform crossover, and 3.35 times more than PMX crossover. This means that, for the Choice Function, low-level heuristics with 2-point crossover performed better overall during the search. On the other hand, HITO-MAB gave too much emphasis to the exploration of the search space. This resulted in it selecting all low-level heuristics almost the same number of times, with a slight preference to low-level heuristics with 2-point crossover (approximately 1.2 times more than uniform crossover and 1.18 times more than PMX crossover). This made HITO-MAB perform close to HITO-R. Of the other systems used in previous work [7], HITO-MAB behaved more similarly to HITO-CF in terms of low-level heuristic selection and obtained similar results overall.

Fig. 1 depicts Pareto fronts of two objectives of the four algorithms for Guava. Each Pareto front has been generated by composing all the non-dominated solutions found in 30 independent executions. As we are minimising for both objectives, the lower the values, the better that front is. HITO-CF yields a Pareto front that dominates the approximation sets of all of the other algorithms. Even though NSGA-II obtained worse hypervolume results than HITO-R, its Pareto front only lacks diversity when compared to HITO-R. NSGA-II's Pareto front dominates almost half of HITO-R's front, whereas HITO-R could not find solutions that dominate any solution found by NSGA-II. This can be explained by the fact that hypervolume not only considers convergence, but also takes into account diversity in its computation.

As a 'sanity check', we also executed a Random Search algorithm, however it performed so poorly that the Pareto fronts of the other algorithms were unreadable when plotted on a graph. As a result we have omitted this algorithm from this section.

5 Conclusion

In this paper we applied a set of selection hyper-heuristics to the ITO problem for Google Guava. The Google Guava system and the number of unit dependencies it contains are larger than the systems previously used in the literature for this problem. The results obtained using hyper-heuristics for the Google Guava instance are coherent with previous results presented in the literature. The best variant, HITO-CF, was able to outperform other versions of HITO and also a well-known NSGA-II. This can be used as evidence that HITO-CF is capable of solving bigger and unseen instances of the ITO problem. Furthermore, we believe

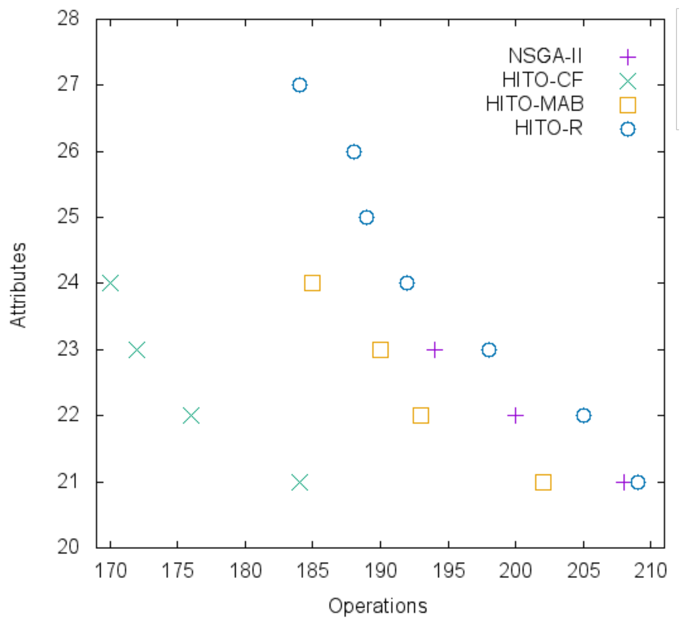

Fig. 1. Pareto fronts found after the 30 independent runs.

that this highlights the suitability of hyper-heuristics for further research in the field of Search Based Software Engineering (SBSE).

References

1. google/guava: Google Core Libraries for Java. https://github.com/google/guava, Accessed: 25-04-2017
2. Assunção, W.K.G., Colanzi, T.E., Vergilio, S.R., Pozo, A.: A multi-objective optimization approach for the integration and test order problem. Information Sciences 267, 119–139 (2014)
3. Coello, C.A.C., Lamont, G.B., Veldhuizen, D.A.V.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer Science, 2nd edn. (2007)
4. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Proceedings of PATAT 2000. pp. 176–190. Springer (2000)
5. Guizzo, G., Fritsche, G.M., Vergilio, S.R., Pozo, A.T.R.: A hyper-heuristic for the multi-objective integration and test order problem. In: Proceedings of GECCO 2015. pp. 1343–1350. ACM (2015)
6. Guizzo, G., Vergilio, S.R., Pozo, A.T.: Evaluating a multi-objective hyper-heuristic for the integration and test order problem. In: 2015 Brazilian Conference on Intelligent Systems (BRACIS). pp. 1–6. IEEE (2015)
7. Guizzo, G., Vergilio, S.R., Pozo, A.T., Fritsche, G.M.: A multi-objective and evolutionary hyper-heuristic applied to the integration and test order problem. Applied Soft Computing 56, 331–344 (2017)
8. Scitools: Understand. https://scitools.com/features/, Accessed: 25-04-2017