MISTA 2009

Multilevel Search for Evolving the Acceptance Criteria of a Hyper-Heuristic

Matthew Hyde · Ender Özcan · Edmund K. Burke

1 Introduction

A hyper-heuristic can be defined as a heuristic which searches a space of heuristics, as opposed to directly searching a solution space [1]. A traditional hyper-heuristic framework therefore consists of two levels. The hyper-heuristic itself operates on the higher level, choosing between a set of problem specific heuristics on the lower level, either with a fixed strategy or with a learning mechanism [2,3].

Perturbative hyper-heuristics use a set of perturbative low level heuristics. They can be defined both by their heuristic selection mechanism, and by their move acceptance criteria. The heuristic selection mechanism iteratively chooses which heuristic to apply, given the current problem state. The acceptance criteria decides whether to accept a move generated by this heuristic. For example, [4] presents a tabu search selection mechanism, with an 'accept all' acceptance criteria. This selection mechanism could easily be combined with an 'accept only improving' acceptance criteria, or a stochastic criteria such as simulated annealing.

Given a fixed heuristic selection mechanism, different acceptance criteria work well on different problems. We show that genetic programming [5] can evolve an effective move acceptance criteria for two problem domains, bin packing [6] and SAT [7,8]. Metaheuristics are popular search techniques for timetabling and scheduling problems, and this technique could be used to automatically design a metaheuristic for a given problem domain.

Genetic programming has been shown to be effective as a hyper-heuristic to generate heuristics over a range of different problem domains, including production scheduling [8–10]. In this case, we will use a similar system to generate the acceptance criteria component of a simple random hyper-heuristic. The genetic programming is therefore operating at a higher level than a hyper-heuristic in a traditional two level framework. The genetic programming is searching a space of hyper-heuristics, not a space of solutions or even problem specific heuristics. It has the characteristics of a more general *multilevel* search methodology, which takes advantage of the recursive nature of the hyper-heuristic definition, 'heuristics which search a space of heuristics'. We present the

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK. E-mail: [mvh,exo,ekb]@cs.nott.ac.uk

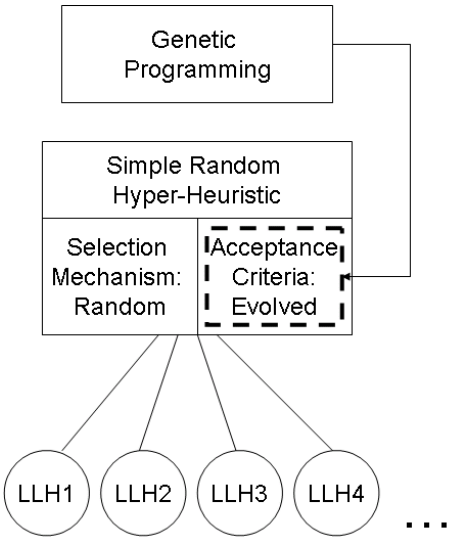

Fig. 1 A multilevel search framework with genetic programming

preliminary results of applying genetic programming as the highest level heuristic in a multilevel search framework, applied to the bin packing and SAT problem domains.

2 Methodology

The simple random heuristic selection mechanism is kept constant, and we compare the performance of both an 'accept all' and a great deluge acceptance criteria, with an acceptance criteria evolved with genetic programming. Our aim is to show that genetic programming is capable of evolving an acceptance criteria which is specialised to a given problem. These experiments are motivated by the observation that 'accept only improving' is more a effective hyper-heuristic acceptance criteria for bin packing than for SAT, while the converse is true for an 'accept all' strategy.

Figure 1 shows the overall structure of the framework. The genetic programming evolves a population of acceptance criteria to be used in the simple random hyper-heuristic. To assess the fitness of an acceptance criteria within the genetic programming algorithm, the simple random hyper-heuristic is run on a problem instance with the acceptance criteria. The hyper-heuristic repeatedly chooses a low-level heuristic (LLH's in figure 1) to apply, which is applied to the current solution. The new solution is accepted if the GP individual returns a value less than 0.5 when evaluated. It is important to note that even improving solutions are accepted based on the result of the GP individual. The hyper-heuristic is asked to solve five instances, and the sum of their objective function values is assigned to the individual as its fitness value.

2.1 Implementation Details

The low-level heuristics that we have implemented for these two problems are well known problem specific heuristics from the literature. Independent sets of low-level

Terminals	Functions
Delta (change in fitness)	+
Current Fitness	−
Previous Fitness	∗
Current Time	% (Protected Divide)
Total Time	e^x

Table 1 The functions and terminals included in the genetic programming system

heuristics are used for bin packing and SAT, and there are seven implemented for bin packing and eight implemented for SAT. The heuristics are all perturbative heuristics, they start with a feasible solution and return a new feasible solution.

The 100 problem instances included for bin packing contain 100 pieces, uniformly distributed between 150 and 200. The bin capacity is set to 1000. These instances can be found at http://paginas.fe.up.pt/∼esicup/tiki-index.php. The 100 problem instances included for SAT are the 'uf250-1065' instances available at http://www.satlib.org/, and they are all satisfiable.

2.2 Genetic Programming Parameters

The acceptance criteria are represented as tree structures, as in canonical genetic programming. The population size is set to 500, and each run lasts for 40 generations. The possible components of the trees (the functions and terminals) are shown in table 1. Protected divide (%) changes a division by zero into a division by 0.001. The fifth function is Euler's number raised to the power of the result of the child node, and as such this is the only function with just one child node. Delta is the change in fitness from the last time that a solution was accepted. The current time is the elapsed time, in milliseconds, since the individual began solving the instance. The total time is effectively a constant of 5000, as each hyper-heuristic individual is allowed to run for five seconds.

3 Preliminary Results

A simple random strategy is employed as the heuristic choice mechanism in all experiments. We use 100 instances from the literature for each problem, and we use five to train a heuristic. 95 instances are therefore used for testing, and the average result of these is reported.

Table 2 shows a summary of the time taken to evolve one heuristic. The fitness of a heuristic is the sum of its results on the five training instances. Each heuristic is run for five seconds, on each of five instances, so it takes 25 seconds to give a fitness score to a heuristic in the population. We evolved 30 heuristics in parallel for each of the two problem domains.

Table 3 presents the results. The figures shown represent the average result over the 95 instances in the test set. For bin packing, the evaluation function is shown in equation 1, where: n = number of bins, $fullness_i$ = sum of all the pieces in bin i, and C = bin capacity. This function returns a value between zero and one, where lower is

Population	500
Generations	40
Instances	5
Time per Instance	5 secs
Total Seconds	500000
Total Hours	138.9

Table 2 The time taken to run the experiments

	Bin Packing	SAT
SR-AM	0.133	5.158
SR-GD	0.078	5.147
Best Evolved	0.063	4.316
Evolved Average	0.065	5.595

Table 3 Comparison of evolved versus fixed acceptance criteria, a lower result is better

better, and a set of completely full bins would return a value of zero.

$$Fitness = 1 - \left(\frac{\sum_{i=1}^{n} (fullness_i/C)^2}{n} \right) \quad (1)$$

The evaluation function for the SAT problem is simply the number of broken clauses. Therefore, a lower value is better, and a value of zero represents a satisfied formula.

Results are shown for the best heuristic and for the average of the 30 heuristics that were evolved. They show that for both the bin packing and SAT problem domains, genetic programming is able to evolve an acceptance criteria that works better than either a great deluge or 'accept all' strategy.

References

1. Ross, P.: Hyper-heuristics. In: E.K. Burke, G. Kendall (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pp. 529–556. Kluwer, Boston (2005)
2. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis **12**(1), 3–23 (2008)
3. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (eds.) Handbook of Meta-Heuristics, pp. 457–474. Kluwer, Boston, Massachusetts (2003)
4. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. Journal of Heuristics **9**(6), 451–470 (2003)
5. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. The MIT Press, Boston, Massachusetts (1992)
6. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons, Chichester (1990)
7. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92), pp. 440–446. San Jose, CA, USA (1992)
8. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation (MIT Press) **16**(1), 31–1 (2008)
9. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In: Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 1559–1565. London, UK. (2007)
10. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling **9**(1), 7–34 (2006)