Evolving Heuristics with Genetic Programming

Mohamed BaderElDen
Department of Computing and Electronic Systems
University of Essex
Wivenhoe Park, United Kingdom
mbbade@essex.ac.uk

Riccardo Poli
Department of Computing and Electronic Systems
University of Essex
Wivenhoe Park, United Kingdom
rpoli@essex.ac.uk

ABSTRACT

Hyper-Heuristics are methods to choose and combine heuristics to generate new ones. In this work, we use a grammar-based genetic programming system as a Hyper-Heuristic framework. The framework is used for evolving effective incremental solvers for SAT (Inc*). Tests against well-known local search heuristics on a variety of benchmark problems reveal that the evolved heuristics are superior.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Experimentation.

Keywords

Genetic Programming, Hyper-Heuristic, Inc*, SAT, Heuristics.

1. INTRODUCTION

Heuristic methods have contributed to the solution of many combinatorial optimisation problems such as bin packing, the travelling salesman problem, graph colouring, and the satisfiability problem (SAT). The performance of heuristics on a problem varies from instance to instance. Also, even on the same instance, randomised heuristics may be able to provide good solutions on one occasion, and bad on another. Hyper-heuristics (HHs) aim to provide a more robust approach raising the level of generality at which optimisation methods operate. They can be defined as "heuristics to choose heuristics" [4]. The main idea is to make use of different heuristic during the search for a solution.

The target in SAT is to determine whether it is possible to set the variables of a given Boolean expression in such a way to make the expression true. The expression is said to be satisfiable if such an assignment exists. If the expression is satisfiable, we often want to know the assignment that satisfies it. The expression is typically represented in Conjunctive Normal Form (CNF), i.e., as a conjunction of clauses, where each clause is a disjunction of variables or negated variables.

Stochastic local-search heuristics have been widely used since the early 90s for solving the SAT problem following the successes of GSAT [6]. The main idea behind these heuristics is to try to get an educated guess as to which variable will most likely, when flipped, give us a solution or to move us one step closer to a solution. Normally the heuristic starts by randomly initialising all the variables in the CNF formula. It then flips one variable at a time, until either a solution is reached or the maximum number of flips allowed has been exceeded.

2. EVOLVING INC* SAT HEURISTICS

SAT is one of the most studied combinatorial optimisation problems, and the first problem proved to be NP-Complete. In this work we use genetic programming (GP) [5] in a HH framework to solve SAT problems. In particular, we use GP to evolve local search heuristics to be used within the Inc* algorithm. Inc* [3] is a general algorithm that can be used in conjunction with any local search heuristic and that has the potential to substantially improve the overall performance of the heuristic. The general idea of the algorithm is the following. Rather than attempting to directly solve a difficult problem, the algorithm dynamically chooses a simplified instance of the problem, and tries to solve it. If the instance is solved, Inc* increases the size of the instance, and repeats this process until the full size of the problem is reached. The search is not restarted when a new instance is presented to the solver. Thus, the solver is progressively biased towards areas of the search space where there is a higher chance of finding a solution to the original problem.

Encouraged by the success of Inc*, here we take Inc* one step further and evolve complete Inc*-type SAT heuristics, instead of just evolving one or two control elements for the human-designed version of Inc*. To do this, we use an extended version of the grammar-based Hyper-Heuristic GP framework (GP-HH) we developed in [2, 1], the grammar is extended and modified to make it more suitable for evolving effective Inc* heuristics. As one can easily see inspecting standard local search heuristics, all the heuristics share similar components, such as: variable score, selection of a clause and conditional branching. By giving GP-HH the freedom to design completely new Inc*-type strategies, we hope to find novel and even more powerful algorithms for the solution of the SAT problem than Inc* or GP-HH alone.

3. EXPERIMENTAL SETUP

In evolving Inc* SAT heuristics we used a training set including 50 SAT problems with different numbers of variables. The

Table 1: Comparison between average performance of WalkSat and WalkSat with Inc* and Inc* with the evolved heuristic (IncHH) SR=success rate, AT = average tries, AF=average number of flips

		WalkSat		IncWalk			IncHH		
name	no. clauses	SR	AF	SR	AF	AT	SR	AF	AT
uf20	91	1	104.43	1	136.32	1.13	1	98.54	0.96
uf50	218	1	673.17	1	702.52	4.25	1	723.52	3.13
uf75	325	1	1896.74	1	1970.59	8.15	1	1909.61	7.17
uf100	430	1	3747.32	1	3640.62	10.31	1	3769.42	9.07
uf150	645	0.97	15021.3	1	13526	15.44	1	6454.14	12.60
uf200	860	0.9	26639.2	0.92	27586.2	20.59	1	26340.8	19.09
uf225	960	0.87	29868.5	0.87	32258.8	21.27	1	34187.7	20.24
uf250	1065	0.81	38972.4	0.83	39303.5	25.15	0.93	39025.6	24.37

problems were taken from the widely used SATLIB benchmark library. All problems were randomly generated satisfiable instances of 3-SAT. In total we used 50 instances: 10 with 100 variables, 15 with 150 variables and 25 with 250 variables. While strategies are evolved using 50 fitness cases, the generality of best of run individuals is then evaluated on an independent test set of SatLib.

In these experiments we used a population of 500 individuals. The GP system initialises the population by randomly drawing nodes from the function and terminal sets. This is done uniformly at random using the GROW method, except that the selection of the function *Flip* is forced for the root node and is not allowed elsewhere. The reproduction rate is 0.1. Individuals that have not affected by any genetic operator are not evaluated again to reduce the computation cost. The crossover rate is 0.8. Offspring are created using a specialised form of crossover. A random crossover point is selected in the first parent, then the grammar is used to select the crossover point from the second parent. It is randomly selected from all valid crossover points. If no point is available, the process is repeated again from the beginning until crossover is successful. Mutation is applied with a rate of 0.1. This is done by selecting a random node from the parent (including the root of the tree), deleting the sub-tree rooted there, and then regenerating it randomly as in the initialisation phase.

For the Inc* algorithm we allowed 1000 flips to start with. Upon failure, the number of flips is incremented by 20%. We allow a maximum total number of flips of 100,000. We evolved Inc* SAT heuristics for one simple Inc* strategies which adds 15% of the of the total number of clauses after each success and remove 10% after each failure.

4. RESULTS

We start by showing a typical example of the Inc* heuristics evolved using the GP Hyper-Heuristics framework. Figure 1 shows one of the best performing heuristics evolved for the Inc* strategy description of the grammar basic components is presented in [2]. As one can see evolved heuristics are significantly more complicated than the standard heuristics we started from (e.g., GSat, WalkSat, Novelty). So, a manual analysis of how the component steps of an evolved heuristic contribute to its overall performance is difficult.

Table 1 shows the results of a set of experiments comparing the performance of the following algorithms: WalkSat alone, WalkSat with the Inc* (IncWalk) and the GPHH evolved heuristic with the Inc* (IncHH). IncHH outperforms the other algorithms.

5. CONCLUSIONS

We have used the GP-HH framework for evolving customised SAT heuristics which is used within the Inc* algorithm. GP has been able to evolve heuristics with high performance on different benchmark SAT problems.

6. ACKNOWLEDGMENTS

The authors acknowledge financial support from EPSRC (grants EP/C523377/1 and EP/C523385/1).

7. REFERENCES

[1] M. B. Bader-El-Den and R. Poli. A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1749–1749, London, 7-11 July 2007. ACM Press.

[2] M. B. Bader-El-Din and R. Poli. Generating SAT local-search heuristics using a GP hyper-heuristic framework. *Proceedings of the 8th International Conference on Artificial Evolution*, 36(1):141–152, 2007.

[3] M. B. Bader-El-Din and R. Poli. Inc*: An incremental approach to improving local search heuristics. In *EvoCOP 2008*. Springer, March 2008. (to appear).

[4] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer Academic Publishers, 2003.

[5] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published by `lulu.com`. Freely available at `http://www.gp-field-guide.org.uk`, 2008.

[6] B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In P. Rosenbloom and P. Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

```
Flip ( ifv(30, If v ( NotZeroAge ,
MacScr(, ifl(20, AllUC, UC), TieRand),
Ifv (40, ScndMacScr (
Ifl(Small, AllUC, UC), TieAge),
Ifv(ZeroBreak, UC, MaxScr(AllUC, TieAge) ) ),
Ifv(90, If v ( NotMinAge , MacScr(UC, TieRand),
If (70, ScndMacScr(UC, TieAge) , Rand(UC) ),
Ifv (ZeroBreak, Ifl(Small, AllUC, UC),
Ifv (40, ifl(20, AllUC, UC), Rand(UC)))
```

Figure 1: Best evolved heuristics for Inc* SAT