Evolving Variable-Ordering Heuristics for Constrained Optimisation

Stuart Bain, John Thornton, and Abdul Sattar

Institute for Integrated and Intelligent Systems
Griffith University
PMB 50, GCMC 9726, Australia
{s.bain, j.thornton, a.sattar}@griffith.edu.au

Abstract. In this paper we present and evaluate an evolutionary approach for learning new constraint satisfaction algorithms, specifically for MAX-SAT optimisation problems. Our approach offers two significant advantages over existing methods: it allows the evolution of more complex combinations of heuristics, and; it can identify fruitful synergies among heuristics. Using four different classes of MAX-SAT problems, we experimentally demonstrate that algorithms evolved with this method exhibit superior performance in comparison to general purpose methods.

1 Introduction

Algorithms to solve MAX-SAT problems encounter a number of additional challenges to regular satisfiability testing: firstly, unless the optimal cost is known *a priori*, a local search is unable to recognise the optimality of a solution. Secondly, for a complete search to prove optimality, the search space of a MAX-SAT problem must be thoroughly examined, being unable to terminate once a satisfying solution has been found. Additionally, until the current cost bound is exceeded, backtracking search must overlook constraint violations that would have triggered immediate backtracking in satisfiability testing.

To overcome these challenges recent work [1, 2] has adopted a two-phase approach, using a greedy local search routine to determine an initial cost bound for a branch and bound procedure, which then determines the globally optimal solution. Each of these works relies on a single ordering heuristic that has been demonstrated to perform well on a generalised range of benchmark instances. However, as good performance on such instances is not necessarily indicative of superior performance on other specific problems [3], how should the algorithm most suited to a specific problem of interest be identified?

Adaptive problem solving methods [4] have been developed to address this and are able to modify their behaviour to suit specific problems. Such methods permit efficient problem specific algorithms to be developed automatically without the involvement of human problem solving expertise. The contribution of this paper is a method by which new algorithms can be evolved for particular classes of problems. In an empirical study, we show that our evolved algorithms significantly outperform existing approaches on a range of NP-hard MAX-SAT optimisation problems.

2 Existing Adaptive Methods

All of the methods considered here adapt by combining in different ways atomic *measures*, i.e. simple functions that describe the nature of the problem and the state of the search.

The MULTI-TAC system developed by Minton [4] is designed to synthesise algorithms for solving CSPs. Exploration of new algorithms is by way of a beam search, designed to control the number of candidate heuristics that will be examined. As the beam search selects only the best B candidate algorithms for further consideration, MULTI-TAC is susceptible to overlooking *synergies*, i.e. measures that perform poorly individually but well in conjunction with other methods.

The Adaptive Constraint Engine (ACE) of Epstein et al. [5] learns the appropriate importance of individual advisors (measures) for particular problems. ACE is only applicable for use with complete search, as a trace of the expanded search tree is necessary to update advisor weights. Although described as being applicable to over-constrained problems, it does not obviously follow how this type of weight update scheme would apply when every search path eventually derives an inconsistency. There appears to be a practical limitation on the complexity of algorithms that can be learned by ACE, but unlike the beam search method used in MULTI-TAC, the use of feedback to update weights facilitates the identification of synergies between heuristics.

A third approach is the CLASS system developed by Fukunaga [6], which can construct algorithms of arbitrary complexity. Adaptation in CLASS is evolutionary, using a specialised composition operator to generate new algorithms. This operator is solely applicable to algorithms of an *if-then* form and, as the offspring generated with it are always larger and more complex than their parents, term rewriting must be used to reduce the size of generated algorithms.

ACE, MULTI-TAC and CLASS each have different strengths, but clearly the potential exists to overcome a number of their limitations. The foregoing discussion has identified a number of features crucial to the expressiveness and performance of an adaptive system:

1. Ability to represent both complete and local search routines
2. Unrestricted complexity
3. Ability to recognise and exploit synergies
4. Appropriateness for satisfiable and over-constrained problems
5. The ability to learn from failure

Our adaptive system exhibits all of these characteristics and is presented in the next section.

3 Evolving Algorithms

As genetic programming [7] has been developed specifically to address the problem of evolving complex structures, it is suprising that it is yet to be successfully applied to the domain of adapting algorithms. This study sets out to redress this absence from the adaptive constraint algorithm literature, but also to study the importance of more complex, non-linear (multiplicative) combinations of heuristics that some previous works have used in only a limited fashion.

A constraint satisfaction algorithm may be viewed as a procedure that iteratively makes *moves*, i.e. variable-value assignments (in complete search), or reassignments (in local search). At each iteration, procedures of both types rank potential *moves* according to heuristic merit. Such a heuristic is simply a functional expression composed from measures describing the nature of the problem and the state of the search. Such a representation satisfies criteria 1 & 2 above, being suitable for search procedures of both types and without an *a priori* complexity bound. One method suitable for the adaptation of these functional expressions is genetic programming.

Genetic programming [7] begins with a random population of expressions, which in this case represent search heuristics. Methods analogous to natural selection and biological reproduction are used to breed subsequent populations of heuristics that better solve the target problem. As the probability of incorporating an individual into the next generation is determined probabilistically by its fitness, poorly performing algorithms are not automatically excluded, permitting synergies to be identified (criteria 3). Furthermore, the fitness measure can incorporate a variety of performance data, making genetic programming suitable for both satisfiable or over-constrained problems (criteria 4) and allowing it to distinguish between heuristics that fail to locate a solution (criteria 5).

3.1 Empirical Study

To determine the best algorithm for each particular training instance (listed in Table 1), the evolutionary procedure was run 5 times and for 50 generations for each instance. The initially generation of algorithms all include the MOMS heuristic but are otherwise randomly generated. The fitness measure used was the number of backtracks necessary to determine the optimal solution to the training instance, standardised so that fewer backtracks equates to higher fitness. The composition of each successive generation was as follows, to give a total of $n_p = 50$ in each generation: the previous $n_c = 3$ best algorithms; $n_b = 36$ new algorithms generated by standard GP crossover; and, $n_m = 11$ algorithms generated by mutation.

The results of the experiments are tabulated in Table 1, both for linear (weighted-sum combinations) and non-linear (multiplicative) combinations of measures. Every evolved algorithm required fewer backtracks than MOMS on its training instance with algoritms employing linear combinations of heuristics offering mean and median improvements over standard MOMS of 56.2% and 62.7% respectively, but algorithms employing non-linear combinations offering mean and median improvements of 58.4% and 65.3% respectively. There was less distinction in terms of time however, with evolved algorithms offering on average no more than a 34% improvement over MOMS.

On its own though, the ability of an algorithm to perform well on a single training problem is not particularly useful, due to the computational time required for training. To be truly useful, good performance on a training instance

Instance	Cost (GSAT / Optimal)	MOMS BTs	Evolved Linear BTs	Evolved Linear Time %MOMS	With Non-linear BTs	With Non-linear Time %MOMS
uuf100-0420.cnf	(2/2)	11030	8571	123.8%	8580	134.3%
MOMS+Degree-3*(RevJW+Linear)						
uuf100-04.cnf	(2/2)	11085	**8491**	132.0%	8706	116.9%
MOMS+CountSatisfy+10*(ValUsed-10*FwdDegree)						
uuf100-0327.cnf	(3/1)	17950	748	8.64%	**472**	4.3%
MOMS+MOMSLiteral*(2SJW-NumWillDetermine)						
uuf100-0190.cnf	(3/2)	31064	7524	42.5%	**5969**	35.2%
MOMS+2SidedJW*(Linear*MOMSStrict+1)+MOMSStrict						
uuf100-0332.cnf	(3/2)	46709	6015	25.5%	**5378**	23.1%
MOMS+1stOrder+MOMS*(MOMSStrict+NumWillDetermine+RevJW)						
p_2_50_200.cnf	(16/16)	5835	**783**	23.2%	798	21.6%
MOMS+20*(100*FwdDegree+2SidedJW)						
p_2_50_250.cnf	(22/22)	27610	5827	35.5%	**4855**	31.8%
MOMS+(Undetermined+UnitClause)*(JeroslowWang*1stOrder*FwdDegree)						
p_2_100_300.cnf	(15/15)	84062	6619	15.4%	**6521**	14.1%
MOMS+(72*Undetermined^2*FwdDegree*Degree)						
jnh310.cnf	(3/3)	4744	3923	89.2%	**3711**	122.4%
MOMS+Degree+(UndetCount*FwdDegree*MOMSStrict*NumConstraints)						
jnh307.cnf	(3/3)	5244	**2668**	68.1%	3177	100.0%
MOMS+FwdDegree-180*Determined						
jnh303.cnf	(3/3)	21554	18102	132.3%	**15830**	103.9%
MOMS+BwdDegree+(BwdDegree*MOMSLiteral*UnitClause*Linear)						
jnh302.cnf	(4/4)	35335	29458	122.7%	**25440**	137.1%
MOMS+MOMSStrict*(UnitClause*UndetCount*MOMSLiteral+1)						
jnh305.cnf	(4/3)	39104	7984	29.5%	**7498**	37.3%
MOMS+CountSatisfy*(MOMSStrict-1stOrder*MOMSLiteral)						
qg3-05.cnf	(5/5)	21935	11817	79.1%	**10998**	71.3%
MOMS+(JeroslowWang*UnitClause+ValUsed)*(MOMSStrict*Linear)						

Table 1. Comparison of performance of evolved algorithms on training instances, along with the preference expression of the best performing algorithm for each. Boldface denotes the algorithm requiring the fewest backtracks.

Class	Num. Instances	MOMS BTs		Linear BTs		Non-Linear BTs	
		Mean	Median	Mean	Median	Mean	Median
uuf100	100	1311	295	**1308**	288	1320	289
jnh	34	3592	550	2665	457	**2325**	475
quasigroup	3	2.95E6	311986	238950	132901	**69037**	84530
MAX-2-SAT	6	3.16E6	629613	410180	87362	**100070**	34628

Table 2. Performance of MOMS and evolved linear & non-linear variants on test sets. Boldface denotes the algorithm requiring fewest mean backtracks.

must translate into good performance on a class of similar problems. To demonstrate that evolved algorithms exhibit such performance, the evolved algorithms for each training instances were evaluated on a larger test set of problems of their class. Performance results for MOMS and the best evolved algorithm in each class are tabulated in Table 2. From these results it can be seen that evolved algorithms, particularly the non-linear variants, offer significant mean and median performance benefits on the larger test sets as well.

4 Conclusions and Future Work

This work has demonstrated a new method for automatically adapting algorithms that exhibits a number of desirable characteristics absent in other work, specifically the ability to discover synergies between heuristics and to explore complex non-linear combinations of heuristics. These two important features are a step toward developing fully automated constraint solving algorithms.

The evolved algorithms were shown to outperform the well-known MOMS heuristic on training instances taken from four different classes of NP-hard optimisation problems. An evaluation of the evolved algorithms on larger test sets of problems showed that on three of the four classes examined, an evolved algorithm substantially outperformed MOMS in terms of backtracks. These results indicate that genetic methods are certainly appropriate for the adaptation of algorithms.

Finally, in evaluating the importance of non-linear algorithms, we found them to have better backtrack performance on 10 of the 14 training instances. Although exhibiting only comparable performance on the *uuf100* test set, non-linear variants achieved superior performance on all other test sets, including both a structured and a smaller random problem set. This suggests that non-linear combinations identify and exploit synergies overlooked by a linear approach, and whilst not appropriate for all problem classes, there can be substantial performance gains from non-linear combinations on problem classes that are sufficiently homogenous.

An extended version of this paper is available from the author's homepage http://stuart.multics.org

References

1. Borchers, B., Furman, J.: A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Journal of Combinatorial Optimization **2** (1999) 299–306
2. Xing, Z., Zhang, W.: Efficient strategies for (weighted) maximum satisfiability. In: CP '04: Principles and Practice of Constraint Programming. (2004) 690–705
3. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
4. Minton, S.: Automatically configuring constraint satisfaction programs: A case study. Constraints **1** (1996) 7–43
5. Epstein, S.L., Freuder, E.C., Wallace, R., Morozov, A., Samuels, B.: The adaptive constraint engine. In: CP '02 Principles and Practice of Constraint Programming. (2002) 525–540
6. Fukunaga, A.: Automated discovery of composite SAT variable-selection heuristics. In: AAAI'02, Canada (2002) 641–648
7. Koza, J.: Genetic Programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, Massachusetts (1992)