Evolution of Neural Networks Topologies and Learning Parameters to Produce Hyper-heuristics for Constraint Satisfaction Problems

José Carlos Ortiz-Bayliss
Tecnológico de Monterrey
Monterrey, Mexico
jcobayliss@gmail.com

Hugo Terashima-Marín
Tecnológico de Monterrey
Monterrey, Mexico
terashima@itesm.mx

Peter Ross
University of Napier
Edinburgh, Scotlant
pross@blueyonder.co.uk

Santiago Enrique Conant-Pablos
Tecnológico de Monterrey
Monterrey, Mexico
sconant@itesm.mx

ABSTRACT

This paper describes a model which constructs hyper-heuristics for variable ordering within Constraint Satisfaction Problems (CSPs) by running a genetic algorithm that evolves the topology of neural networks and some learning parameters.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Constraint Satisfaction, Genetic Algorithms, Neural Networks, Hyper-heuristics

1. INTRODUCTION

A Constraint Satisfaction Problem (CSP) is defined by a set of variables X, where each variable is associated a domain D of values subject to a set of constraints C [6]. The goal is to find a consistent assignment of values to variables in such a way that all constraints are satisfied, or to show that a consistent assignment does not exist. There are many theoretical and practical applications of CSPs (see for example [3, 4]) and these problems are usually solved by using a depth first search.

2. SOLUTION APPROACH

Three variable ordering heuristics were selected to be used in this investigation: MXC, SD and $E(N)$ (for details see [1, 2]. The basic idea behind the proposed model is that, given a certain instance, a neural network has to decide which variable ordering heuristic to use at each node of the search tree. Every time a variable is instantiated, a new subproblem arises and the properties may differ from the previous instance. The idea is to solve the problem by constructing the answer, deciding which heuristic to apply at each step.

The way to generate this neural network involves an evolutionary process that determines the best architecture and learning parameters for the task. The networks used for this research are backpropagation neural networks with a sigmoidal transference function. Also, we have incorporated the momentum to our networks to improve their performance. Each neural network deals with a simplified problem state described by the constraint density (p_1) and tightness (p_2) and uses them as input values. The output of the network is the heuristic to apply at a given time.

Even when it is possible to generate a neural network and train it without the need of any evolutionary process, it is not clear the learning parameters and the topology of the network that should be used to maximise the quality of the network. One of the ideas to find the right topology (number of neurons in first and second hidden layers) and learning parameters (learning rate, momentum and ages) is to use a genetic algorithm to determine these values. A steady-state genetic algorithm runs until a trained neural network is obtained. The topology of the network and the learning parameters are fully determined by the evolutionary process.

3. EXPERIMENTS AND RESULTS

Before applying any neural network approach it is necessary to design the pattern that will be used for training. We decided to use a training pattern that maps every point in the space (p_2, p_1) to one of the three heuristics MXC, SD or $E(N)$. To obtain the pattern we produced a grid of instances in the range $[0, 1]$ with increments of 0.025 in both dimensions. For each point in the grid we generated 30 random instances and the heuristic with the lower average consistency checks was selected as the best option. Thus, we produced and analysed a grid containing 50430 instances to obtain the training pattern. At the end, we obtained a

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

Table 1: Percentage of instances where the hyper-heuristics reduce the average number of consistency checks required by each low-level heuristic

HH	MXC	SD	$E(N)$
NHH06	50.395%	33.636%	9.375%
NHH13	50.353%	32.027%	9.395%
NHH09	50.306%	31.759%	9.812%
NHH01	47.752%	32.201%	9.446%
NHH03	46.851%	32.437%	9.049%

Table 2: Performance of the hyper-heuristics for Testing Set

HH	better $< 98\%$	equal $98 - 102\%$	not as good $> 115\%$	W
NEHH04	0.661%	77.355%	21.983%	78.017%
NEHH01	0.661%	77.355%	21.983%	78.017%
NEHH03	0.826%	76.198%	22.975%	77.025%
NEHH05	0.992%	75.702%	23.305%	76.694%
NEHH02	0.331%	76.198%	23.471%	76.529%
AVG	0.694%	76.562%	22.743%	77.256%

matrix that represents a 'rule' that indicates which heuristic to apply given the properties p_1 and p_2.

Applying the same concept of grids of instances, we generated two sets: one for training and other for testing. At each point in the grid, five random instances were generated by using increments of 0.1 in p_1 and p_2. The Training and Testing Set were generated using 20 variables and 10 values in their domains. The Training Set was used during the genetic algorithm process to evaluate the fitness of the hyper-heuristics and the Testing Set was used only for testing purposes and was never used during the genetic algorithm process. Five runs of the genetic algorithm were conducted using 30 individuals, 100 cycles, crossover probability = 1.0 and mutation rate = 0.1. At the end of each run, the best individual of the last population was selected as the resulting hyper-heuristic. Thus, we produced five hyper-heuristics which later were tested and compared against the results of the low-level heuristics on the Testing Set.

First of all we are interested in showing the benefit of using any of our hyper-heuristics instead of the low-level heuristics. Table 1 presents the average percentage of reduction in the number of consistency checks achieved by each one of the five hyper-heuristics with respect to the low-level heuristics when tested on all instances in both sets. The results show that the hyper-heuristics are able to reduce the number of consistency checks required by MXC, SD and $E(N)$ in all the cases. $E(N)$ seems to be a very good heuristic and even when the hyper-heuristics are able to overcome the average performance of this heuristic, the improvement is not as large as for the other heuristics. These results support the idea that these hyper-heuristics provide a general method for solving a wide range of instances with acceptable results. Moreover, when compared with each single heuristic, the hyper-heuristics overcome the performance of the low-level heuristics.

In a more challenging comparison, we measured the quality of our hyper-heuristics when compared against the best low-level heuristic. The higher the value of the sum of first two columns (which we will call W), the better the performance of the hyper-heuristic. These results are presented in Table 2. The average value of W for the Testing Set is above 75%, which means that at least for 3/4 of the instances the hyper-heuristics behave at least as well as the best low-level heuristic and not worse. Even thought NEHH01 and NEHH04 have the higher average value of W, they are not able to overcome the best result of the low-level heuristics in most of the cases. Actually, the produced hyper-heuristics seem not to be able to beat the best of the three low-level heuristics when they work on the same instances and the best result is kept. We need to be clear in the fact that W does not measure the proportion of instances where the hyper-heuristic is better than the best result of the low-level heuristics but the proportion of instances where the hyper-heuristic is at least as good as the best low-level heuristic.

4. CONCLUSIONS AND FUTURE WORK

We have presented a model which consists of a genetic algorithm which evolves the topologies and learning parameters of neural networks that represent hyper-heuristics. The hyper-heuristics produced are very competitive in terms of consistency checks. One immediate consideration for future work will be to include heuristics for value ordering in our model and conduct the proper experiments to test its implications. We also want to extend our results for real instances like timetabling or scheduling represented as CSP to determine its real contribution. Also, a deeper analysis about the relation between generation effort and quality is required. Is the quality of the solutions worth the effort to generate a hyper-heuristic? At this moment we strongly believe it does, but a more detailed study is needed to confirm this idea.

5. ACKNOWLEDGMENTS

This research was supported in part by ITESM under the Research Chair CAT-144 and the CONACYT Project under grant 99695.

6. REFERENCES

[1] D. Brelaz. New methods to colour the vertices of a graph. *Communications of the ACM*, 22, 1979.

[2] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T.Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of CP-96*, pages 179–193, 1996.

[3] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[4] U. Montanari. Networks of constraints: fundamentals properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[5] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín. Mapping the performance of heuristics for constraint satisfaction. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, july 2010.

[6] C. P. Williams and T. Hogg. Using deep structure to locate hard problems. In *Proc. of AAAI-92*, pages 472–477, San Jose, CA, 1992.