A Neuro-Evolutionary Approach to Produce General Hyper-heuristics for the Dynamic Variable Ordering in Hard Binary Constraint Satisfaction Problems

José Carlos Ortiz-Bayliss, Hugo Terashima-Marín
ITESM-Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
{a00796625, terashima}@itesm.mx

Peter Ross
School of Computing
Napier University
Edinburgh EH10 5DT UK
P.Ross@napier.ac.uk

Jorge Iván Fuentes-Rosado, Manuel Valenzuela-Rendón
ITESM-Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
jivfur@gmail.com, valenzuela@itesm.mx

ABSTRACT

This paper introduces a neuro-evolutionary approach to produce hyper-heuristics for the dynamic variable ordering for hard binary constraint satisfaction problems. The model uses a GA to evolve a population of neural networks architectures and parameters. For every cycle in the GA process, the new networks are trained using backpropagation. When the process is over, the best trained individual in the last population of neural networks represents the general hyper-heuristic.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods and Search*

General Terms

Algorithms

Keywords

Neuro-Evolutionary Computation, Hyper-heuristics, Optimization, Constraint Satisfaction Problems, Dynamic Variable Ordering

1. INTRODUCTION

A Constraint Satisfaction Problem (CSP) is defined by a set of variables and a set of constraints. Each variable has a nonempty domain of possible values. Each constraint involves some subset of variables and specifies the allowable combinations of values for that subset [3]. Many researchers have proved, based on analysis and experimentation, the importance of the variable ordering and its impact in the cost of the solution search [2]. This ordering has repercussions in the complexity of the search, and this can be done either in static or in dynamic fashion. This investigation deals with the dynamic variable ordering, where the order is constructed during the search, based on some criteria about the characteristics of the variables left to instantiate. It has been proved in many studies that dynamic ordering gives better results than static ordering [1].

This paper presents a method to produce a general hyper-heuristic intended to provide a way to order variables for a wide variety of instances of CSP, so they can be solved. The procedure learns the hyper-heuristic by going through a training phase using instances with a variety of features. The generated hyper-heuristic is tested later with a collection of seen and unseen examples providing acceptable results. The general method is based on the evolution of neural networks, where the inputs of the networks are a series of condition-action rules obtained through an evolutionary approach from a previous investigation. [4].

GECCO'09, July 8–12, 2009, Montréal, Québec, Canada.
ACM 978-1-60558-325-9/09/07.

2. SOLUTION APPROACH

The solution model used in this research is an extension of the work described at. [4]. The previous approach produced hyper-heuristics composed by series of blocks, where each block can be seen as a condition-action rule. We used the rules obtained from the previous approach as inputs for the proposed model to obtain a more robust and general hyper-heuristic. This hyper-heuristic is coded in a neural network where the inputs are the state of the current problem and the output is the low-level heuristic to apply for that problem state. What is to be achieved in this research is the use of a GA to find the architecture and values for the neural network parameters by treating the network as a function, and optimizing the architecture and parameters. Then, the evolved and trained neural network is able to combine single heuristics for variable ordering to solve efficiently a wide variety of instances of CSP.

The parameters of the networks that are considered for evolution in the GA are: neurons in the first hidden layer $[1, 15]$, neurons in the second hidden layer $[0, 15]$, momentum for back propagation $[0.0, 1.0]$, ages for the backpropagation training $[444, 1000]$, learning rate $[0.1, 0.8]$, minimum and maximum weight of the synapse $[-3, +3]$.

The neural network deals with a simplified problem state space and uses it as input values. The problem state is formed by eight real numbers (inputs to the neural network) that describe the current problem state. The problem state representation includes the next criteria: percentage of variables that remain to be instantiated (v), percentage of

values in all the domains left to instantiate (d), percentage of remaining constraints in the problem (c_j), percentage of large, medium and small constraints left in the problem (lc, mc, sc), and finally the Rho and Kappa factors, respectively (r, k).

The neuro-evolutionary process consists of the following steps: (1) Generate the initial population, (2) Train the individuals using backpropagation, (3) Evaluate each individual, (4) Apply selection, crossover, mutation and replacement, and (5) Repeat from step 2 until the termination criterion is reached.

Every individual in the last population from five runs of the model described in [4] was decomposed to extract individual blocks, which were used to construct the training table for the neural networks in our model. The resulting training table consisted of 687 records.

To compute the fitness for each chromosome, the distance between the solution obtained by that individual (HH_i) with respect to the best result given by the single heuristic (BH_i) is measured. The current cycle in the evolutionary process is also taken into consideration. The fitness is an average given by:

$$f(x) = \frac{\sum_i^{5+c} \frac{BH_i}{HH_i}}{5+c} \quad (1)$$

where BH_i is the best result obtained through the set of low-level heuristics for instance i, HH_i is the consistency checks required by the hyper-heuristic to solve instance i and finally, c represents the cycle in the evolutionary process. This evaluation guarantees that, the greater the number of $f(x)$, the better the quality of the hyper-heuristic.

2.1 Description of the instances

The instances are divided into three main groups: one training set and two testing sets. The general procedure consists of solving first all instances in the three sets with the single heuristics. This is carried out to keep the best solution that is later used also by the model we propose. The next step is to let the process work on the training set until the termination criterion is reached and a general hyper-heuristic is produced. All instances in the sets are then solved with this general hyper-heuristic.

2.2 Results

We randomly generated 1250 hard instances using the algorithm proposed by Prosser [2], divided as follows: 600 for the training set, 400 for testing set I and 250 for training set II. Both the training and the testing set I are composed by distinct instances with $n = 20$ and $m = \{10, 20\}$. Training set II contains instances with $n = 30$ and $m = 10$. Note that instances in training set II are used only for testing, meaning that they are unseen examples.

We used a steady stable GA with tournament selection of size two, crossover and mutation rate of 1.0 and 0.1, respectively. The population size was set to 30 individuals and the GA run over 200 cycles. the best network from the last population was selected as the hyper-heuristic (NEHH). The performance of each low-level heuristic and the NEHH were compared with the best result from the low-level heuristics when used to solve the instances in the testing sets. The results are shown in Table 1.

As we can observe, even when the hyper-heuristic is very competitive, it is overcome just by the *kappa* heuristic, which presents the best results for 73% of the instances. For 1.44% of the instances, NEHH is able to outperform the best low-level heuristic by reducing in more than 15% the number of consistency checks needed to solve the instances.

Table 1: Low-level heuristics and NEHH compared with the best result of the low-level heuristics for the testing sets.

	Reduction		Equal	Increment	
Method	> 15%	15% to 3%	± 3%	3% to 15%	> 15%
Kappa	0.00	0.00	73.00	10.00	17.00
NEHH	1.44	0.00	69.96%	9.43	19.14
E(N)	0.00	0.00	31.00	28.67	40.33
FF	0.00	0.00	6.67	27.00	66.33
Rho	0.00	0.00	5.67	23.67	70.66
MC	0.00	0.00	2.67	0.33	97.00
MFF	0.00	0.00	1.33	0.67	98.00
Bz	0.00	0.00	0.00	18.33	81.67

3. CONCLUSIONS

This document has described a model based on neuro-evolution which evolves architectures of neural networks by using a GA and trains every network through backpropagation. The networks are applied selection, crossover and mutation operators in order to improve their fitness at each cycle of the evolutionary process. The best individual in the last population of networks is taken as the general hyper-heuristic. This hyper-heuristic takes advantage of the characteristics of the problem and tries to apply the more suitable low-level heuristic for that problem state. Overall, the process identifies general hyper-heuristics after going through a learning procedure with training and testing phases.

4. ACKNOWLEDGMENTS

This research was supported in part by ITESM under the Research Chair CAT-010 and the CONACYT Project under grant 41515.

5. REFERENCES

[1] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 38(2):211–242, 1994.

[2] P. Prosser. Binary constraint satisfaction problems: Some are harder than others. In *Proceedings of the European Conference in Artificial Intelligence*, pages 95–99, Amsterdam, Holland, 1994.

[3] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 1995.

[4] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross and M. Valenzuela-Rendón. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 571–578. Atlanta, Georgia, USA, 2008