Neural Networks to Guide the Selection of Heuristics within Constraint Satisfaction Problems

José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos

Tecnológico de Monterrey, Campus Monterrey
Monterrey, Mexico, 64849
jcobayliss@gmail.com, terashima@itesm.mx, sconant@itesm.mx

Abstract. Hyper-heuristics are methodologies used to choose from a set of heuristics and decide which one to apply given some properties of the current instance. When solving a Constraint Satisfaction Problem, the order in which the variables are selected to be instantiated has implications in the complexity of the search. We propose a neural network hyper-heuristic approach for variable ordering within Constraint Satisfaction Problems. The first step in our approach requires to generate a pattern that maps any given instance, expressed in terms of constraint density and tightness, to one adequate heuristic. That pattern is later used to train various neural networks which represent hyper-heuristics. The results suggest that neural networks generated through this methodology represent a feasible alternative to code hyper-heuristic which exploit the strengths of the heuristics to minimise the cost of finding a solution.

Keywords: Constraint Satisfaction, 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 [31]. 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. CSPs belong to the NP-Complete class [10] and there is a wide range of theoretical and practical applications like scheduling, timetabling, cutting stock, planning, machine vision, temporal reasoning, among others (see for example [9], [14], [17]).

Several deterministic methods to solve CSPs exist [18,29], and solutions are found by searching systematically through the possible assignments to variables, guided by heuristics. It is a common practice to use Depth First Search (DFS) to solve CSPs [26]. When using DFS to solve CSPs, every variable represents a node in the tree and the deeper we go in the tree, the larger the number of variables that have already been assigned a feasible value. Every time a variable is instantiated, a consistency check occurs to verify that the current assignment does not conflict with any of the previous assignments given the constraints within the instance. When an assignment produces a conflict with one or more constraints, the instantiation must be undone, and a new value must be assigned to that variable. When the feasible values decrease to zero, the value of a previously instantiated variable must be changed, this is known as backtracking [2]. Backtracking always goes up one single level in the search tree when a backward move is needed. Backjumping is another powerful technique for retracting and modifying the value of a previously instantiated variable and goes up more levels than backtracking in the search tree. Another way to reduce the search space is using constraint propagation, where the idea is to propagate the effect of one instantiation to the rest of the variables due to the constraints among the variables. Thus, every time a variable is instantiated, the values of the other variables that are not allowed due to the current instantiation are removed.

The general idea in this investigation is to combine the strengths of some existing heuristics to generate a method that chooses among them based on the features of the current instance. Hyper-heuristics are methods that choose from a set of heuristics and decide which one to apply given some properties of the instances. Because of this, they seem to be a suitable technique to implement our idea. Different approaches have been used to generate hyper-heuristics (see for example: [1], [4] and [21]) and they have proven to achieve promising results for many optimization problems such as scheduling, transportation, packing and allocation.

This paper is organized as follows. Section 2 presents a brief description of previous studies related to this research. Section 3 describes the methodology used in our solution model. The experiments and main results are presented in Sect. 4. Finally, Sect. 5 presents the conclusions and future work.

2 Background and Related Work

When using neural networks to solve CSPs, it is common to convert the CSP into an optimization problem, where the task of the network is to minimise a non-negative function that vanishes only for solutions [13]. Tsang and Wang [30] described a neural network approach called GENET for solving CSPs with binary constraints through a convergence procedure. Nakano and Nagamatu [19] proposed a Lagrange neural network for solving CSPs where, in addition to the constraints, each CSP has an objective function. Jönsson and Södenberg developed a neural network approach for solving boolean CSPs and later the same approach was extended to more general CSPs [13].

Even though the term hyper-heuristic was first introduced by Denzinger et al. [7] in 1997, the idea of combining heuristics goes back to 1960s ([8], [6]). Surveys on hyper-heuristic methodologies can be found in [4], [24], and [5]. One of the first attempts to systematically map CSPs to algorithms and heuristics according to the features of the problems was presented in [29]. In that study, the authors presented a survey of algorithms and heuristics for solving CSPs and they proposed a relation between the formulation of the CSP and the most adequate solving method for that formulation. More recently, Ortiz-Bayliss et al. [20] developed a study about heuristics for variable ordering within CSPs and a way to exploit their different behaviours to construct hyper-heuristics by using a static decision matrix to select the heuristic to apply given the current state of the problem. More studies about hyper-heuristics applied to CSPs include the work done by Terashima-Marín et al. [28], who proposed an evolutionary framework to generate hyper-heuristics for variable ordering in CSPs; and Bittle and Fox [3] who presented a hyper-heuristic approach for variable and value ordering for CSPs based on a symbolic cognitive architecture augmented with case based reasoning as the machine learning mechanism for their hyper-heuristics. The difference between these two approaches lies in the learning method and the set of heuristics used.

3 Solution Model

This section presents the proposed solution model in detail. It describes the problem state representation, the neural network and the way in which the networks are trained and used to code the hyper-heuristics.

3.1 Problem State Representation

For this research we have included only binary CSPs. A binary CSP contains unitary and binary constraints only. Rossi et al. [25] proved that for every general CSP there is an equivalent binary CSP. Thus, all general CSPs can be reduced into a binary CSP. To represent the problem state we propose to use two important binary CSPs properties known as constraint density (p_1) and constraint tightness (p_2). The constraint density is a measure of the proportion of constraints within the instance; the closer the value of p_1 to 1, the larger the number of constraints in the instance. A value of $p_1 = 0.5$ indicates that half of the nodes present a constraint among them. The constraint tightness (p_2) represents a proportion of the conflicts within the constraints. A conflict is a pair of values $\langle x, y \rangle$ that is not allowed for two variables at the same time. The higher the number of conflicts, the more unlikely an instance has a solution. A CSP instance with $p_2 = 1$ is trivially insoluble because all pairs of values in the constraints are not allowed. In contrast, an instance with $p_2 = 0$ does not contain any conflicts and can be solved very easily because all the pairs of values between variables are allowed. We used these two measures to represent the problem state. Our idea is that these two features can be used to describe a CSP instance and to create a relation between instances and heuristics.

The CSP instances used for this research are randomly generated in two stages. In the first stage, a constraint graph G with n nodes is randomly constructed and then, in the second stage, the incompatibility graph C is formed by randomly selecting a set of edges (incompatible pairs of values) for each edge (constraint) in G. The instance generator receives four parameters: $\langle n, m, p_1, p_2 \rangle$. The number of variables is defined by n and the uniform domain size by m. The parameter p_1 determines how many constraints exist in a CSP instance and it is called constraint density, whereas p_2 determines how restrictive the constraints are and it is called constraint tightness. More details on the framework for problem instance generation can be found in [22] and [27].

Every time a variable is assigned a new value and the infeasible values are removed from domains of the remaining uninstantiated variables, the values of p_1 and p_2 change and a sub-problem with new features appears. This is the reason why we decided to use the constraint density and tightness to represent the problem state and guide the selection of the low-level heuristics. Our idea is that these two features can be used to describe a CSP instance and to create a relation between instances and heuristics.

3.2 Variable Ordering Heuristics

A solution to any given CSP is constructed selecting one variable at the time based on one of the four variable ordering heuristics used in this investigation: Rho, Max-Conflicts (MXC), Minimum Remaining Values (MRV) and Expected Number of Solutions ($E(N)$). Each one of these heuristics orders the variables to be instantiated dynamically at each step during the search process. These heuristics are briefly explained in the following lines.

– The Rho heuristic is based on the approximated calculation of the solution density ρ. This measure considers that, if a constraint C_i prohibits in average a fraction p_c of possible assignations, a fraction $1 - p_c$ of assignations is allowed. Then, the average solution density ρ is the average fraction of allowed assignations through all the constraints. If independence between the constraints is supposed, then ρ is defined as [11]:

$$\rho = \prod_{c \in C} (1 - p_c). \tag{1}$$

  The basic idea with the Rho heuristic is to select the variable that enters in the subproblem which contains the largest fraction of solution states. This is, the subproblem with the largest solution density.

– MXC selects the variable that is involved in the larger number of conflicts among the constraints in the instance. The assignment will produce a subproblem that minimises the number of conflicts among the uninstantiated variables.

– MRV is one of the most simple and effective heuristics to determine which variable to instantiate [12,23]. This heuristic selects the variable with the less number of available values in its domain. The idea consists basically in taking the most restricted variable from those which have not been instantiated yet and by doing so, reducing the branching factor of the search.

– $E(N)$ selects the variable in such a way that the subproblem maximizes the expected number of solutions [11]. This heuristic will maximize the size of the subproblem so as the solution density. The value of $E(N)$ is calculated as:

$$E(N) = \prod_{x \in X} |D_x| \times \rho. \tag{2}$$

We have also used Min-Conflicts [16,15] as value ordering heuristic to improve the search. The selection of the value, when using Min-Conflicts, prefers the value involved in the minimum number of conflicts [15]. This heuristic will try to leave the maximum flexibility for subsequent variable assignments. Min-Conflicts is not considered in the selection process of heuristics because it is a value ordering heuristic. In this investigation, Min-Conflicts is used as a complement of the four variable ordering heuristics to improve the overall performance of the model.

3.3 The Training Set

Before applying any neural network approach it is necessary to design the pattern that will be used for training. If the training set is wrong, then we will produce networks which are not useful for the problem. We decide to use a training set that maps every point in the space (p_2, p_1) to one of the four heuristics previously explained. To obtain this set we produced a grid of instances in the range $[0, 1]$ with increments of 0.025 in each dimension. For every point in the grid we generated 30 random instances which were solved using each of the four heuristics. The heuristic with the lower average consistency checks was selected as the best heuristic for those coordinates. Thus, we produced and analysed a grid containing 50430 instances to obtain the training set. The discussed grid represents a 'rule' that indicates which heuristic to apply given the properties p_1 and p_2. The set obtained via this methodology is shown in Fig. 1. In this figure, the best heuristic for each point in the grid (in terms of average consistency checks), is shown. The training set allows us to observe the regions on the space where each heuristic is more suitable than the others. The points in the grid with no mark indicate that there was not a significant difference in the means of the consistency checks of two or more heuristics.

3.4 Neural Networks to Represent Hyper-heuristics

The basic idea behind the proposed hyper-heuristics 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 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

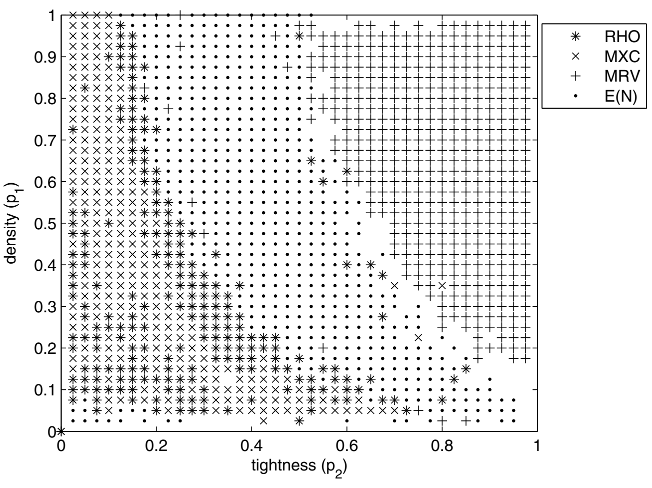

Fig. 1. The training set used for the neural networks

network deals with a simplified problem state described by p_1 and p_2 and uses them as input values; the output of the network is the heuristic to apply at a given time. Once the neural network has been trained using the set from Sect. 3.3 it represents a complete recipe for solving a problem, using a simple algorithm.

Until the problem is solved: (a) determine the current problem state, (b) use the neural network to decide which heuristic to apply, (c) apply the heuristic attached to the state and (d) update the state. This process is shown in Fig. 2.

4 Experiments and Results

The testing set includes 1000 different random instances generated with $n = 20$ and $m = 10$. The instances are uniformly distributed in the space $p_1 \times p_2$, with 10 instances per point. This set forms a grid of instances with increments of 0.1 in each axis, starting from instances with $(p_2 = 0.1, p_1 = 0.1)$ up to instances with $(p_2 = 1, p_1 = 1)$.

We obtained 20 hyper-heuristics as the result of running 20 times the solution model explained in Sect. 3, and those hyper-heuristics were tested with all the instances in the Testing Set. All the networks were generated with two input neurons (p_1 and p_2), two hidden layers and four neurons in the output layer (one for each heuristic). The number of neurons in each hidden layer was randomly decided at the moment the network was created and lied within the range $[5, 15]$. The learning rate and momentum were also randomly selected at generation time. We decided to randomize these parameters to obtain networks with different topologies and observe the differences in the results.

The 20 hyper-heuristics were compared with the average result of the simple heuristics in terms of consistency checks. A consistency check occurs every time

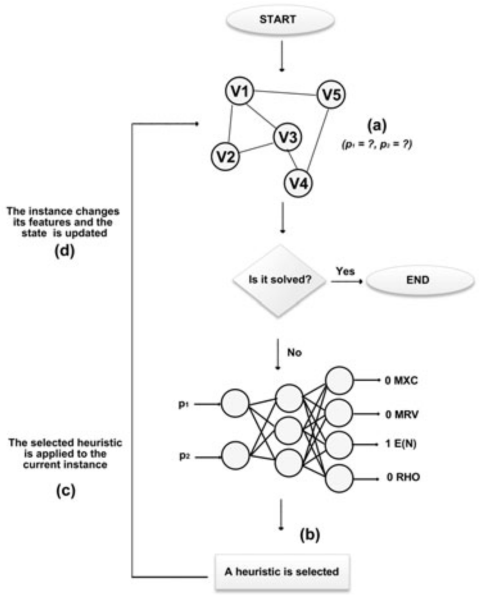
START
V1
V2
V3
V4
V5
(a)
(p₁ = ?, p₂ = ?)
The instance changes its features and the state is updated
(d)
Is it solved?
Yes
END
No
p₁
p₂
0 MXC
0 MRV
1 E(N)
0 RHO
The selected heuristic is applied to the current instance
(c)
(b)
A heuristic is selected

Fig. 2. Process of applying the hyper-heuristic

a constraint must be verified and it is a common measure of the quality of the CSP solving methods. The results of the performance of the hyper-heuristics are shown in Table 1. In this table, $W(mean)$ is a the proportion of instances where the hyper-heuristic performs at least as well as the mean result of the simple heuristics. For the cases where the hyper-heuristic is better than the mean result of the simple heuristics, $W(mean)$ does not provide any information about the percentage of the reduction of consistency checks. This information is presented with $R(mean)$, which indicates the mean reduction in the number of consistency checks with respect to the mean result of the heuristics for every instance in the testing set.

The results suggest that the hyper-heuristics produced with our model represent a feasible solution method for solving CSPs. We have proven that any of these hyper-heuristics will behave at least as well as the mean result of the simple heuristics for a large proportion of instances (no less than 77.2%). In the case of NEHH17, the maximum value of R is achieved with 87.1%. In terms of reduction, NHH08 is the better choice because it is able to reduce the consistency checks of the mean result of the heuristics in 85.3%, in average, for every instance in the testing set.

As an additional result, we tested NHH08 and NHH17 against the best result of the simple heuristics. NHH08 and NHH17 obtained values of $W(Best)$ of 63.0% and 65.8%, respectively. The decrease in the value of W when compared

Table 1. Performance of the hyper-heuristics when compared against the mean result of the simple heuristics

HH	$W(mean)$	$R(mean)$	HH	$W(mean)$	$R(mean)$
NHH01	78.800%	17.777%	NHH11	78.200%	16.086%
NHH02	85.900%	23.739%	NHH12	86.700%	23.522%
NHH03	87.300%	25.195%	NHH13	77.200%	14.331%
NHH04	78.800%	15.148%	NHH14	79.900%	18.234%
NHH05	79.400%	17.627%	NHH15	85.900%	23.534%
NHH06	77.300%	13.246%	NHH16	86.700%	23.371%
NHH07	80.400%	16.471%	**NHH17**	**87.100%**	24.540%
NHH08	85.300%	**25.257%**	NHH18	77.700%	13.851%
NHH09	80.200%	17.939%	NHH19	85.600%	21.188%
NHH10	79.000%	16.547%	NHH20	86.300%	23.729%

with the best result of the heuristics is not necessarily a bad result. In this case we can identify the best result because we are using a small set of heuristics and random instances with suitable generation parameters, but in practice it is not feasible to try various heuristics for each instance and keep the best result. Our hyper-heuristics are not able to overcome the best heuristic for all the cases but they provide acceptable results for a wide range of instances.

5 Conclusions and Future Work

We have presented a methodology to obtain information from a set of instances to produce a pattern that matches CSPs to heuristics. We used that pattern to produce backpropagation neural networks that decide which heuristic to apply given the features of the instances. These neural networks represent variable ordering hyper-heuristics for CSPs and obtained promising results when compared against the mean result of the simple heuristics. Even when these hyper-heuristics need more work to improve their performance, the preliminary results suggest that neural networks hyper-heuristics provide a feasible method for solving CSPs.

As future work we are interested in adding value ordering heuristics to the selection process of the hyper-heuristic and see its contribution to the performance of the model. We also think it is important to test our approach on real instances. Finally, more work is needed to understand the patterns of heuristics and produce new ways to exploit those patterns to guide the heuristic selection.

Acknowledgments

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

References

1. Bilgin, B., Özcan, E., Korkmaz, E.E.: An experimental study on hyper-heuristics and exam timetabling. In: Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling, pp. 123–140 (2006)
2. Bitner, J.R., Reingold, E.M.: Backtrack programming techniques. Commun. ACM 18, 651–656 (1975)
3. Bittle, S.A., Fox, M.S.: Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, GECCO 2009, pp. 2209–2212. ACM Press, New York (2009)
4. Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Shulenburg, S.: Hyper-heuristics: an emerging direction in modern research technology. In: Handbook of metaheuristics, pp. 457–474. Kluwer Academic Publishers, Dordrecht (2003)
5. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
6. Crowston, W.B., Glover, F., Thompson, G.L., Trawick, J.D.: Probabilistic and parametric learning combinations of local job shop scheduling rules, p. 117 (1963)
7. Denzinger, J., Fuchs, M., Fuchs, M., Informatik, F.F., Munchen, T.: High performance atp systems by combining several ai methods. In: Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997), pp. 102–107. Morgan Kaufmann, San Francisco (1997)
8. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference, Carnegie Institute of Technology (1961)
9. Freuder, E.C., Mackworth, A.K.: Constraint-Based Reasoning. MIT/Elsevier, Cambridge (1994)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
11. Gent, I., MacIntyre, E., Prosser, P., Smith, B.: T.Walsh.: An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In: Proceedings of CP 1996, pp. 179–193 (1996)
12. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14, 263–313 (1980)
13. Jönsson, H., Söderberg, B.: An information-based neural approach to generic constraint satisfaction. Artificial Intelligence 142(1), 1–17 (2002)
14. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence 8(1), 99–118 (1977)
15. Minton, S., Johnston, M.D., Phillips, A., Laird, P.: Minimizing conflicts: A heuristic repair method for csp and scheduling problems. Artificial Intellgence 58, 161–205 (1992)
16. Minton, S., Phillips, A., Laird, P.: Solving large-scale csp and scheduling problems using a heuristic repair method. In: Proceedings of the 8th AAAI Conference, pp. 17–24 (1990)
17. Montanari, U.: Networks of constraints: fundamentals properties and applications to picture processing. Information Sciences 7, 95–132 (1974)
18. Nadel, B.A.: Algorithms for constraint satisfaction: a survey. AI Magazine 13(1), 32–44 (1992)

19. Nakano, T., Nagamatu, M.: Lagrange neural network for solving csp which includes linear inequality constraints. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3697, pp. 943–948. Springer, Heidelberg (2005)
20. Ortiz-Bayliss, J.C., Özcan, E., Parkes, A.J., Terashima-Marín, H.: Mapping the performance of heuristics for constraint satisfaction. In: IEEE Congress on Evolutionary Computation (CEC 2010), pp. 1–8 (July 2010)
21. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligence Data Analysis 12(1), 3–23 (2008)
22. Prosser, P.: An empirical study of phase transitions in binary constraint satisfaction problems. Tech. Rep. Report AISL-49-94, University of Strathclyde (1994)
23. Purdom, P.W.: Search rearrangement backtracking and polynomial average time. Artificial Intelligence 21, 117–133 (1983)
24. Ross, P., Marfn-Blazquez, J.: Constructive hyper-heuristics in class timetabling, vol. 2 (September 2005)
25. Rossi, F., Petrie, C., Dhar, V.: On the equivalence of constraint satisfaction problems. In: Proceedings of the 9th European Conference on Artificial Intelligence, pp. 550–556 (1990)
26. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
27. Smith, B.M.: Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence 81, 155–181 (1996)
28. Terashima-Marín, H., Ross, P., Farías-Zárate, C., López-Camacho, E., Valenzuela-Rendón, M.: Generalized hyper-heuristics for solving 2d regular and irregular packing problems. Annals of Operations Research 179, 369–392 (2010)
29. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press Limited, London (1993)
30. Tsang, E.P.K., Wang, C.J.: A generic neural network approach for constraint satisfaction problems. In: Neural Network Applications, pp. 12–22. Springer, Heidelberg (1992)
31. Williams, C.P., Hogg, T.: Using deep structure to locate hard problems. In: Proc. of AAAI 1992, San Jose, CA, pp. 472–477 (1992)