On the Idea of Evolving Decision Matrix Hyper-Heuristics for Solving 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

Ender Özcan
School of Computer Science
University of Nottingham
Nottingham, United Kingdom
exo@cs.nott.ac.uk

Andrew J. Parkes
School of Computer Science
University of Nottingham
Nottingham, United Kingdom
ajp@cs.nott.ac.uk

ABSTRACT

When solving a Constraint Satisfaction Problem (CSP), the order in which the variables are selected to be instantiated has implications in the complexity of the search. This work presents the first ideas for evolving hyper-heuristics as decision matrices where the elements in the matrix represent the variable ordering heuristic to apply according to the constraint density and tightness of the current instance.

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, Hyper-heuristics

1. INTRODUCTION

A Constraint Satisfaction Problem (CSP) is defined as follows: given a set of n variables with its respective domain of possible values m and a set of constraints, each one involving a subset of the variables, find all possible n-tuples such that each n-tuple is an instantiation of the n variables satisfying all the constraints. The relevance of studying CSPs lies on the fact that they are an important technique used to find a solution for many artificial intelligence problems [3, 5]. Stated as a classic search problem, a CSP is usually solved using a Depth First Search (DFS) where every variable represents a node in the tree. The selection of the next variable to instantiate affects the search complexity and represents an opportunity to optimize the search. Hyper-heuristics deal with the problem of selecting the best low-level heuristic for a certain problem, given the current characteristics [2].

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

Our model produces hyper-heuristics coded as matrices of integers by going through an evolutionary process which consists of a steady-state genetic algorithm that selects, crosses and mutates the matrices in order to improve their quality. Each matrix represents a rule for the application of the low-level heuristics based on the values of the constraint density (p_1) and tightness (p_2) of the instance at hand. The best decision matrix of the last population in each run is selected as the hyper-heuristic and tested on various instances.

2. SOLUTION APPROACH

A solution for any given CSP is constructed selecting one variable at the time based on one of the two variable ordering heuristics used in this investigation: MXC [6] and SD [1]. Once a variable has been selected, a value must be chosen from its domain and assigned to that variable considering the constraints and using Min-Conflicts [4] as value ordering heuristic.

The collection of instances includes 1210 different instances, divided into two sets: 605 for Set A (used for training within the genetic algorithm) and 605 for Set B (used for testing). Both sets A and B are composed by distinct instances with $n = 20$ and $m = 10$. The instances in the sets are uniformly distributed in the space $p_1 \times p_2$, with five instances per point. Each one of the sets forms a grid of instances with increments of 0.1 in each axis, starting from instances with $(p_2 = 0, p_1 = 0)$ up to instances with $(p_2 = 1, p_1 = 1)$.

A steady-state genetic algorithm is used for evolving the hyper-heuristics until a given number of cycles is achieved. In each cycle, the new individuals (hyper-heuristics) are trained using random instances from the instance Set A. The best individual of the last population of the genetic algorithm is selected to be the resulting hyper-heuristic for that run. Each hyper-heuristic is composed by a matrix of integers. Each matrix contains the description of the problem state representation (the value of x-axis and y-axis which represent p_2 and p_1, respectively) and a low-level heuristic that should be applied given the current problem state under exploration. The hyper-heuristic decides which low-level heuristic to apply by selecting the cell in the matrix with the problem state representation closest to the problem state at hand. Every time a variable is instantiated, the problem

Table 1: Performance of the hyper-heuristics for Set B when compared with the best result of the low-level heuristics

HH	better $< 98\%$	as good $98 - 102\%$	not as good $> 102\%$
MHH02	9.917%	55.041%	35.041%
MHH10	9.752%	51.901%	38.347%
MHH04	9.421%	50.083%	40.496%
MHH09	8.264%	54.050%	37.686%
MHH03	8.099%	49.587%	42.314%
MHH05	7.769%	51.570%	40.661%
MHH01	7.107%	55.372%	37.521%
MHH06	7.107%	53.554%	39.339%
MHH07	6.942%	52.066%	40.992%
MHH08	6.612%	52.397%	40.992%
AVERAGE	8.099%	52.562%	39.339%

Table 2: Comparison between MHH02 and DHH against the best result of the low-level heuristics

HH	better $< 98\%$	as good $98 - 102\%$	not as good $> 102\%$
MHH02	9.917%	55.041%	35.041%
DHH	5.289%	71.736%	22.975%

state is updated and the process is repeated until a solution is found. The algorithm uses a tournament selection of size five and *ad-hoc* crossover and mutation operators. The crossover operator selects a rectangle of cells in the first individual and replaces it with the values of the corresponding cells of the second individual. The mutation operator randomly selects one rectangle of cells from the decision matrix coded within the individual and changes the value of its elements to its complement.

3. EXPERIMENTS AND RESULTS

We ran our model ten times selecting the best hyper-heuristic from each run. Those hyper-heuristics were tested on instances from set B against the best result of the low-level heuristics. The results presented in Table 1 show that the ten hyper-heuristics are able to overcome the best result of the low-level heuristics in at least 6.6% when tested on Set B, and for some cases this value gets close to 10% (MHH02 and MHH10). MHH02 is the best hyper-heuristic because it reduces the number of consistency checks by the larger percentage of instances.

We also decided to test the best hyper-heuristic against one hyper-heuristic taken from the literature. The hyper-heuristic DHH described in [6] was used to solve Set B and the results were compared with those obtained by using MHH02. The results show that DHH obtains the higher percentage of instances where it behaves as well as the best heuristic, but it is not able to reduce the number of consistency checks in proportion to MHH02 (Table 2). DHH seems to be suitable for behaving just as the best result of the low-level heuristics and MHH02 seems to be a better method for combining the use of the heuristics to improve the best result. There is one more issue to be considered in the comparison: the proportion of instances where the method does not perform as well as the best low-level heuristic is higher for MHH02 than for DHH. There is a large difference in the size of the sets used for generating the hyper-heuristics and the way those sets are used to create the hyper-heuristics, and that may be the cause of the difference in the results. These two hyper-heuristics were obtained through different approaches and they both provide promising results.

4. CONCLUSIONS AND FUTURE WORK

The hyper-heuristics were able to map the features of the CSP instances (p_1 and p_2) to a good low-level heuristic and guide the selection of heuristics during the search. When the hyper-heuristics were compared with the best result of the low-level heuristics they performed well but they still need to be improved. Moreover, we must also recall that the best result is not always produced by the same heuristic, and in real cases there is no way to know in advance which heuristic is "the best" for a given instance of the problem. The results confirm the idea behind hyper-heuristics that they are able to exploit the problem-specific features and choose the heuristic which best adapts to that, and achieve a better performance. We also compared one of the hyper-heuristics produced with our approach against DHH, which is another hyper-heuristic taken from the literature and the results show that both methods achieve good results. As future work, we are interested in applying this approach to instances from CSP libraries and real instances. We want to include more heuristics to the model and investigate whether or not this can improve the performance of the system.

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] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Shulenburg. Hyper-heuristics: an emerging direction in modern research technology. In *Handbook of metaheuristics*, pages 457–474. Kluwer Academic Publishers, 2003.

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

[4] S. Minton, M. D. Johnston, A. Phillips, and P. Laird. Minimizing conflicts: A heuristic repair method for csp and scheduling problems. *Artificial Intellgence*, 58:161–205, 1992.

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

[6] 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.