A Supervised Learning Approach to Construct Hyper-heuristics for Constraint Satisfaction

José Carlos Ortiz-Bayliss[1], Hugo Terashima-Marín[2], and Santiago Enrique Conant-Pablos[2]

[1] Automated Scheduling, Optimisation and Planning (ASAP)
School of Computer Science, University of Nottingham, UK
Jose.Ortiz_Bayliss@nottingham.ac.uk
[2] Tecnológico de Monterrey, Campus Monterrey, Mexico
{terashima,sconant}@itesm.mx

Abstract. Hyper-heuristics are methodologies that 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. In this paper we propose a logistic regression model to generate hyper-heuristics for variable ordering within constraint satisfaction problems. The first step in our approach requires to generate a training set that maps any given instance, expressed in terms of some of their features, to one suitable variable ordering heuristic. This set is used later to train the system and generate a hyper-heuristic that decides which heuristic to apply given the current features of the instances at hand at different steps of the search. The results suggest that hyper-heuristics generated through this methodology allow us to exploit the strengths of the heuristics to minimize the cost of the search.

Keywords: Constraint Satisfaction, Hyper-heuristics, Logistic Regression.

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 [17]. 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 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] and [17].

Several deterministic methods to solve CSPs exist(see for example [14], 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 [24]. When using DFS to solve CSPs, every variable represents a node in the tree and the deeper we go in that tree, the larger the

J.A. Carrasco-Ochoa et al. (Eds.): MCPR 2013, LNCS 7914, pp. 284–293, 2013.

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 in 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 [11]. 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 [10]. Thus, every time a variable is instantiated, the values of the other variables that are not allowed due to the current instantiation are removed.

Logistic regression is a type of regression analysis used for classification problems. The idea is to map from a set of input variables to one output variable that represents a class. Logistic regression is one type of supervised machine learning because training examples where the expected output corresponding to the given input must be provided. The general idea of this investigation is to combine the strengths of some existing heuristics through a logistic regression approach to generate a method that chooses among heuristics 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], [5] and [21]) and they have achieved promising results on 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 which includes the features used to characterize the CSP instances, the set of heuristics used and the hyper-heuristic model. The experiments and main results are discussed in Sec. 4. Finally, Sec. 5 presents the conclusion and future work.

2 Background and Related Work

The idea of combining heuristics goes back to 1960s [8] and has been used in many investigations under different names [26,28,7]. Hyper-heuristics are one alternative to combine the strengths of heuristics based on the current problem features. Hyper-heuristics can be divided into two main classes: those which select from existing heuristics and those that generate new heuristics [21]. A more detailed description about the classification of hyper-heuristics can be found

in [6]. In this investigation we will focus our attention on hyper-heuristics that select from existing heuristics.

The first attempts to systematically map CSPs to algorithms and heuristics according to the features of the instances were presented in [29]. In their work, the authors presented a survey of algorithms and heuristics for solving CSPs and proposed a relation between the formulation of the CSP and the most adequate solving method for that formulation. More recently, Ortiz-Bayliss et al. [19] 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. [27], who proposed a genetic algorithm framework to produce hyper-heuristics for variable ordering; Bittle and Fox [3] who presented a hyper-heuristic approach for variable and value ordering based on a symbolic cognitive architecture augmented with case based reasoning as the machine learning mechanism for their hyper-heuristics; and recent works where neural networks are used as hyper-heuristics for variable ordering [18,20]. The differences between all these works on hyper-heuristics for CSPs lies in the set of heuristics used and the learning mechanism used to produce the hyper-heuristics.

3 Solution Model

In this section we discuss the problem state representation, the set of heuristics and the hyper-heuristic model used in this investigation.

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. [23] 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 the use of three important binary CSPs properties known as constraint density (p_1), constraint tightness (p_2) and κ [12]. 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. 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. The value of κ is suggested in the literature as a general measure of how restricted a combinatorial problem is. If κ is small, the instances usually have many solutions with respect to their size. When κ is large, instead, the instances often have few solutions or do not have any at all [12]. κ is defined as $\kappa = \frac{-\sum_{c\in C}\log_2(1-p_c)}{\sum_{x\in X}\log_2(m_x)}$, where p_c is the fraction of unfeasible tuples on constraint c and m_x is the domain size of variable x. It has been found that the most difficult instances with respect to their size occur when $\kappa \approx 1$ [12].

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, p_2 and κ change, and a sub-problem with new features appears. This is the reason why we decided to use this set of features to represent the problem state and guide the selection of the heuristics.

3.2 Variable Ordering Heuristics

A solution to any given CSP is constructed by selecting one variable at the time based on one of the five variable ordering heuristics used in this investigation: minimum domain (DOM), maximum weighted degree (WDEG), domain over weighted degree (DOM/WDEG), kappa (K) and maximum tightness (MXT). 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.

DOM. This heuristic selects the variable with the fewer available values in its domain [13,22].

WDEG. This heuristic attaches a counter, called *weight*, to every constraint of the problem [4,15]. The counters will be updated during the search whenever a dead-end occurs (no more values available for the current variable remain). This heuristic gives priority the variables with the largest weighted degrees.

DOM/WDEG This is a combination of DOM and WDEG heuristics. It selects first the variable that maximizes the quotient of the domain size over the weighted degree of the variable.

K orders the variables based on the value of the kappa factor, κ. K will select first the variable that minimizes the value of κ of the remaining instance [12].

MXT prefers the variable with the tightest constraints (the one with the highest average value of p_2 among all the constraints where it is involved).

We have also used Min-Conflicts [16] as value ordering heuristic to improve the search. When using Min-Conflicts, the next value to try for the selected variable is the one involved in the minimum number of conflicts. Min-Conflicts is not considered as part of the hyper-heuristic model because it is a value ordering heuristic and at the moment we are only using the hyper-heuristic approach for variable ordering. We expect to extend our approach to include value ordering as part of the hyper-heuristic on future developments.

3.3 Instances Used

Our set of instances includes 1000 random binary CSPs distributed among three sets. We will refer to these sets as training set, cross validation set and testing set. These sets contain 600, 200 and 200 instances, respectively. All the CSP instances used for this research were randomly generated with a modified version of model D [25]. First, a constraint graph G with n nodes is randomly constructed and then, 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 five parameters: $\langle n, m, \sigma_m, w_1, w_2 \rangle$. The number of variables is defined by n and the domain size by m; with a maximum deviation of σ in the domain size of each variable. The parameter w_1 determines the probability that a constraint exists in the CSP instance, whereas w_2 determines the probability that an unfeasible pair of values occurs on each constraint.

For each instance, the number of variables was randomly selected in the range $[15, 30]$, where each variable can contain a domain of size in the range $[10, 20]$ (the domain is not uniform among the variables within each instance because of the parameter σ_m used by the generator). The values of w_1 and w_2 were determined by independently choosing at random values in the range $[0, 1]$ for each instance.

Because sometimes more than one heuristic obtains the best result for a given instance (the one with the minimum number of consistency checks), it was necessary to perform a filtering process during the generation of the instances. All the instances where more than one heuristic obtained the best result in terms of consistency checks were discarded and another one was created.

3.4 The Hyper-heuristic Model

This investigation describes a hyper-heuristic model for variable ordering on CSPs based on a logistic regression approach. The hyper-heuristic proposed in this investigation dynamically decides which heuristic to apply as the search progreses. At each step of the search, every time a new variable is to be instantiated, the hyper-heuristic decides which heuristic to apply according to the current problem state (defined by the values of p_1, p_2 and κ).

The hyper-heuristic contains a module for multi-class logistic regression. The hyper-heuristic needs to be trained before being applied. A detailed description of the training process will be provided in the next sections. The core of the hyper-heuristic contains a sigmoid function:

$$h(\boldsymbol{\theta_h}, \boldsymbol{f}) = \frac{1}{1 + e^{\boldsymbol{\theta_h} \cdot \boldsymbol{f}}} \tag{1}$$

where $\boldsymbol{f}$ is the vector of features that characterizes the current problem instance (p_1, p_2, κ). The vector $\boldsymbol{\theta_h}$ is adjusted for each heuristic during the training phase.

In our model, each heuristic is associated a specific vector $\boldsymbol{\theta_h}$. The function $h(\boldsymbol{\theta_h}, \boldsymbol{f})$ is evaluated with the corresponding $\boldsymbol{\theta_h}$ from each heuristic and the vector of features $\boldsymbol{f}$ of the current problem state. The heuristic which $\boldsymbol{\theta_h}$ produces the largest output is selected to be applied on the instance. This is a common way to implement multi-class classification by using logistic regression. In the next section we will discuss how to obtain the vectors $\boldsymbol{\theta_h}$ associated to each heuristic.

4 Experiments and Results

In this investigation, we are using logistic regression for multi-class classification. A vector $\boldsymbol{\theta_h}$ is generated for each heuristic. A set of examples containing the

features $\boldsymbol{f}$ and the best heuristic for such features was obtained from the training set. For each instance in this set, its features (p_1, p_2 and κ) and the most suitable heuristic (the one that required the fewer consistency checks for the search) were saved. The training set was later used to produce specific training examples for each heuristic. These particular training examples contain only binary outputs: 1 when the example corresponds to problem features that made the current heuristic the best option, and 0 otherwise. We used a gradient descend procedure to obtain the values of $\boldsymbol{\theta_h}$ that minimize the cost function for each heuristic. We used 0.01 as learning rate and 1000 iterations to minimize the cost function.

There is a cost function associated to the minimization problem. In this case, the cost function is given by:

$$J(\boldsymbol{\theta_h}) = \begin{cases} \log(h(\boldsymbol{\theta_h}, \boldsymbol{f})) & y = 1 \\ \log(1 - h(\boldsymbol{\theta_h}, \boldsymbol{f})) & y = 0 \end{cases} \tag{2}$$

Then, the idea is to find a vector $\boldsymbol{\theta_h}$ that minimizes the cost function $J(\boldsymbol{\theta_h})$. With gradient descend, at each iteration we must simultaneously update all the values in $\boldsymbol{\theta_h}$ by using the following equation:

$$\boldsymbol{\theta_h} = \boldsymbol{\theta_h} - \alpha \sum_{j=1}^{l} h(\boldsymbol{\theta_h}, \boldsymbol{f}^{(j)}) \boldsymbol{f}_i^{(j)} \tag{3}$$

where l is the number of examples in the training set and $\boldsymbol{f}^{(j)}$ is the jth example of the training set and $\boldsymbol{f}_i^j$ is the feature i of the jth example of the training set (p_1, p_2 and κ, in that order).

At the end, the values of the vectors $\boldsymbol{\theta_h}$ obtained for each heuristic are:

$$\begin{gathered} \boldsymbol{\theta_{MRV}} = [-5.6922, 2.5416, 1.6944, 0.6989] \\ \boldsymbol{\theta_{WDEG}} = [1.7820, -0.6832, 1.8075, -2.8147] \\ \boldsymbol{\theta_{DOM/WDEG}} = [-5.6356, 2.7119, 3.9738, 0.3091] \\ \boldsymbol{\theta_K} = [-3.1786, 1.6747, 2.0354, 0.1376] \\ \boldsymbol{\theta_{MXT}} = [-1.6886, 0.7275, -4.9322, 0.7212] \end{gathered} \tag{4}$$

We used this set of vectors as the core of the hyper-heuristic that was tested in the following experiments. Nevertheless the vector of features $\boldsymbol{f}$ contains only three elements, $\boldsymbol{\theta_h}$ contains four because the first element in the vector corresponds to a bias. To be consistent, a fixed feature with a constant value of 1 is added at the first position of the vector of features $\boldsymbol{f}$. Thus, the first element of the vector $\boldsymbol{\theta_h}$ is always multiplied by 1 at the moment of calculating $\boldsymbol{\theta_h} \cdot \boldsymbol{f}$.

4.1 Evaluating the Hyper-heuristics

We tried the hyper-heuristic obtained with the proposed approach on the three sets. The results are shown in Table 1. It is important to stress that for these instances, all the methods are able to find a solution or to prove that none exists. Then, the only difference is the number of consistency checks which is used for comparison of the quality of the methods.

Table 1. Percentage of instances on each set where each method obtains the best result (requires the fewer consistency checks)

Method	Training set	Validation set	Test set
MRV	5.00%	3.50%	3.50%
WDEG	**57.50**%	**57.00**%	**55.00**%
DOM/WDEG	8.83%	10.50%	7.50%
K	18.83%	20.50%	22.50%
MXT	9.83%	8.50%	11.50%
HH	**62.67**%	**64.50**%	**60.50**%

We can observe that, even though WDEG is clearly the heuristic that achieves the minimum number of consistency checks on the largest fraction of instances on the three sets, the hyper-heuristic is able to overcome this proportion in all the sets. One consideration about these results is the fact that the hyper-heuristic is choosing among different heuristics during the search and then, the cost of the search (in terms of consistency checks) is not exactly the same than the best result obtained by the heuristics applied in isolation. Then, we must interpret the results from table 1 as the proportion of instances where the hyper-heuristic behaves at least as well as the best result obtained with the heuristics applied in isolation.

In Table 2 we present the average costs of each method on the three sets. The average cost for each method is calculated as the average consistency checks required by each method to solve an instance in the set (the sum of the costs on all the instances over the number of instances).

Table 2. Average cost per instance (in consistency checks) for each method on the three sets

Method	Training set	Validation set	Test set
MRV	228430	299319	79901
WDEG	252989	451341	133057
DOM/WDEG	195621	**222157**	84241
K	**160538**	222780	**74548**
MXT	286714	301132	258061
HH	190447	229294	82566

The results of the average costs per instance on the different sets show that, even though WDEG was the heuristic that most of the times obtained the best results, it is not the heuristic with the smallest average cost. This is an interesting result that makes us think that most of the instances that were best solved by using WDEG were not hard, and then, the variance in the results with respect to the other heuristics was small. On the other hand, the instances where K and DOM/DEG (on the cross validation set) were the best options, represent large reductions in the number of consistency checks.

Even though the hyper-heuristic is not able to overcome K with respect to the average cost of the search, it is a very competent solving method with respect to the other heuristics. It is important to stress the difference in the average cost of WDEG and the hyper-heuristic. The percentage of instances where both methods achieved the best results are very close to each other (around 5%), but when we evaluate the average cost, the hyper-heuristic proves its real contribution. The hyper-heuristic is, in all the sets, a best solving method than WDEG. The reductions in the average costs obtained by using the hyper-heuristic, with respect to WDEG, are of 24.72%, 49.19% and 37.94% for the training, cross validation and test set, respectively.

5 Conclusion and Future Work

We have explored the use of a logistic regression approach to produce hyper-heuristics for CSPs. The results show that it is possible to map a CSP instance to one suitable heuristic given the described features. The hyper-heuristic outperforms the best heuristic in the fraction of instances where it achieves the best results. Also, the hyper-heuristic is very competent with respect to the average cost per instance. We observed that the hyper-heuristic is able to exploit the strengths of individual heuristics to perform well on distinct sets of instances.

Even though the results are promising, more work is needed regarding the features used to characterize the instances. In this investigation we used the constraint density (p_1), constraint tightness (p_2) and kappa (κ), but we consider that more features are needed to improve the mapping from instances to heuristics. We think the most important idea to be addressed in the future is the analysis of other relevant CSP features that could lead to a better classification of the instances and the solving methods according to those features.

We also observed that there are opportunities to improve the approach in the way we select the best heuristic for a given instance. In this investigation we produced a set of examples by mapping the instance features to the heuristic that required the fewer consistency checks on that instance. By using this idea we concluded that WDEG was the best heuristic. Nevertheless WDEG was the heuristic with the largest proportion of best results, it was not the heuristic with the minimum average cost per instance. Then, it may be a good idea to explore other alternatives to create the training examples. This is left as part of the future work.

Finally, we are interested in testing our approach on other classes of instances. For example, we would like to apply it to real problems such as scheduling and timetabling, and some optimization problems from vision and biology. This will raise the question of whether one heuristic exists that dominates the others for each specific problem domain.

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: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 394–412. Springer, Heidelberg (2007)
2. Bitner, J.R., Reingold, E.M.: Backtrack programming techniques. Communications of the ACM 18(11), 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, pp. 2209–2212. ACM (2009)
4. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: European Conference on Artificial Intelligence (ECAI 2004), pp. 146–150 (2004)
5. 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 (2003)
6. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research and Management Science, vol. 146, pp. 449–468. Springer (2010)
7. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: A robust optimisation method applied to nurse scheduling. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 851–860. Springer, Heidelberg (2002)
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 (1994)
10. Freuder, E.C.: Synthesizing constraint expressions. Communications of the ACM 21(11), 958–966 (1978)
11. Gaschnig, J.G.: A general backtrack algorithm that eliminates most redundant tests. In: Proceedings of the 5th International Joint Conference on Artificial Intelligence, vol. 1, p. 457. Morgan Kaufmann Publishers (1977)
12. Gent, I., MacIntyre, E., Prosser, P., Smith, B., Wals, T.: An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 179–193. Springer, Heidelberg (1996)
13. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14, 263–313 (1980)
14. Kumar, V.: Algorithms for constraint satisfaction: a survey. AI Magazine 13(1), 32–44 (1992)
15. Lecoutre, C., Boussemart, F., Hemery, F.: Backjump-based techniques versus conflict-directed heuristics. In: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2004, pp. 549–557. IEEE Computer Society, Washington, DC (2004)
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. Ortiz-Bayliss, J., Terashima-Marín, H., Conant-Pablos, S.: Neural networks to guide the selection of heuristics within constraint satisfaction problems. In: Martínez-Trinidad, J., Carrasco-Ochoa, J., Ben-Youssef Brants, C., Hancock, E. (eds.) MCPR 2011. LNCS, vol. 6718, pp. 250–259. Springer, Heidelberg (2011)
19. Ortiz-Bayliss, J.C., Özcan, E., Parkes, A.J., Terashima-Marín, H.: Mapping the performance of heuristics for constraint satisfaction. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010), pp. 1–8. IEEE Press (2010)
20. Ortiz-Bayliss, J.C., Terashima-Marín, H., Conant-Pablos, S.E.: Learning vector quantization for variable ordering in constraint satisfaction problems. Pattern Recogn. Lett. 34(4), 423–432 (2013)
21. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis 12(1), 3–23 (2008)
22. Purdom, P.W.: Search rearrangement backtracking and polynomial average time. Artificial Intelligence 21, 117–133 (1983)
23. 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)
24. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach. Prentice Hall (1995)
25. Smith, B.M.: Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence 81, 155–181 (1996)
26. Storer, R.H., Wu, S.D., Vaccari, R.: New search spaces for sequencing problems with application to job shop scheduling. Management Science 38(10), 1495–1509 (1992)
27. Terashima-Marín, H., Ortiz-Bayliss, J.C., Ross, P., Valenzuela-Rendón, M.: Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO 2008), pp. 571–578. ACM (2008)
28. Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M.: Evolution of constraint satisfaction strategies in examination timetabling. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 635–642. Morgan Kaufmann (1999)
29. Tsang, E., Kwan, A.: Mapping constraint satisfaction problems to algorithms and heuristics. Tech. Rep. CSM-198, Department of Computer Sciences, University of Essex (1993)