Data Mining: an Aid Towards more Efficient Hyper-heuristic Search

Edmund K. Burke, Jingpeng Li* and Rong Qu
{ekb, jpl, rxq}@cs.nott.ac.uk
School of Computer Science
The University of Nottingham
Nottingham, NG8 1BB, United Kingdom

* Corresponding author

1 Introduction

Hyper-heuristics is an emerging search technology motivated by the goal of raising the level of generality at which optimisation systems can work (Burke et al. 2003). A hyper-heuristic can be thought of as a process which uses a higher level search methodology to choose lower level heuristics. Hence, the most distinct feature of a hyper-heuristic lies in its exploration on the search space of heuristics, rather than on the solution space of the problem in hand. As the neighbourhood structures of these two spaces are defined under different domains, this feature enables a hyper-heuristic to be capable of jumping within the solution space of the problem through smooth moves among the neighbourhoods defined by the search space of heuristics.

Compared with problem specific special purpose meta-heuristics, hyper-heuristics should have several features. They should be faster to implement and more flexible to use. Also, they should produce good quality solutions without requiring too much domain knowledge. In hyper-heuristic development there is an obvious trade-off between "generality" and "solution quality".

For the purposes of illustration, consider a dummy timetabling problem of assigning m events to n time slots. A hyper-heuristic uses some method at the higher level to choose from k heuristics at the lower level. According to the way that a solution is built, the hyper-heuristics can be grouped into two classes: constructive and improvement. The former one builds solutions by applying a sequence of heuristics from scratch, while the later one improves solutions by applying one of the heuristics which are often related to the shifting or swapping of the solution components. Clearly, the solution space of the original problem is n^m. For a constructive hyper-heuristic, its solution space is k^m. For an improvement hyper-heuristic, its solution space is $\sum_{j=1}^{t}\sum_{i=1}^{k} f(h_i) = O(tkn^r)$, where $f(h_i)$ is the number of possible moves that the i-th heuristic can take (e.g. $f(h_i) = n(n-1)/2$ if the i-th heuristic is to randomly swap the time slots of two events), $r \in [1, n-1]$ is the maximum number of concurrent shifts allowed in all k heuristics (e.g. $r = 2$ if the i-th heuristic is a swap one), and $t \in [1, n^m - 1]$ is the number of improvements that we wish the hyper-heuristic could make.

Among the above three solution spaces, the one of an improvement hyper-heuristic is the smallest because it normally consists of a r value being smaller than 3 and a t value being fixed, and thus in practice it does not increase exponentially with the problem size. Hence, the improvement hyper-heuristic is the most flexible and it works particularly well if a good initial solution is provided. A constructive hyper-heuristic also operates on a much reduced search space because k is mostly smaller than n. The problem induced by an improvement hyper-heuristic is NP hard in the worst case where $r = n-1$ and $t \to m^n - 1$, while the problem induced by a constructive-heuristic is NP-hard under any circumstance. Hence, the optimal solutions for both problems are normally impossible to find. Even if we find them, they might only correspond to one of the many possible solutions of the original problem and thus we cannot claim that the original problem has been well solved.

To address this weakness, the only solution is allowing more appropriate low level heuristics, as well as more iterations of runs, in the hyper-heuristic search so that each optimal solution (or a number of near optimal solutions at the least) of the original problem has the opportunity to be reached. However, this would significantly increase the search space and consequently increase the computational time since the search by each heuristic is costly, in particular for the constructive heuristics. Hence, so far almost all the research has been concentrated on the development of various clever (or intelligent) systems, such as

refined heuristics, meta-heuristics and AI-based approaches. Those systems act as a supervisor at a higher level to manage a small number heuristics at a low level, with the aim of achieving a balance between the solution quality and the execution time. From a certain point on, this kind of research makes hyper-heuristics not too much different from ordinary optimisation search algorithms, if simply considering each low level heuristic as one of the solution components with a fixed or variable fitness value.

More specifically, since the year 2000 when the term of hyper-heuristics was first coined, there have been at least 70 papers which have appeared in the literature (see http://www.asap.cs.nott.ac.uk/projects/ngds/hhref.shtml). At each decision point within the search space of heuristics, all of the hyper-heuristics presented in these papers implement the following 2-stage computations: first using the chosen heuristic (or sequence of heuristics) to generate a solution, then calculating the object value of the resulting solution.

Intuitively, we feel that there is no need to carry out the costly 2-stage computations for every resulting solution (e.g. for a solution that is obviously infeasible), since many problems in the real world are highly constrained which means that the regions containing feasible solutions are rather scattered and small. In particular, in our previous experiences on two types of nurse rostering problems (Aickelin et al. 2007; Burke et al. 2008), we observed that the close neighbours of an infeasible solution are mostly infeasible, while the close neighbours of a feasible solution are often feasible as well. In that sense, in order to run more iterations on a larger heuristic search space (i.e. containing more heuristics), such a question that has attracted little attention before arises: would it be possible for a hyper-heuristic to suspend the 2-stage computations if it foresees the incoming heuristic (or sequence of heuristics) should not perform well (e.g. cannot generate an infeasible solution)?

2 Data mining by neural network and logistic regression

Data mining (Bozdogan 2004) provides a "yes" answer to the above question. Generally speaking, data mining is the search for relationships and global patterns that exist in large databases but are hidden among the vast amount of data. By learning from examples, data mining is able to archive the goal of how to partition or classify the data, i.e. it formulates classification rules. Hence, without the 2-stage computations, the performance (i.e. the worth of acceptance or rejection) of an incoming heuristic (or sequence of heuristics) can be predicted by applying the classification rules. Compared with the time-consuming computations which are needed for every move in the hyper-heuristic search, the learning of classification rules would be much faster.

In this abstract, we report our preliminary work of applying two fundamentally different data mining techniques, namely the Multi-Layer Perception (MLP) neural network (Rumelhart et al. 1986) and the binary logistic regression, to classify the solutions of a graph-based hyper-heuristic proposed for exam timetabling problems (Burke et al. 2007). In the graph-based hyper-heuristic, a potential solution is represented by a sequence of k heuristics, with each heuristic representing a specific method to decide which exam should be allotted next. Hence, in our prediction models, we can regard the choice of the i-th heuristic as an independent variable xi , $i = 1,..., k$. The "acceptance or rejection" of a resulting sequence can be regarded as a dependent variable y. We anticipate that our models can learn to predict the value of variable y based on the variable values of $(x1,..., xk)$.

We first use the MLP neural network to build our classification model. Basically, an MLP network provides a model of data relationships through highly interconnected and simulated components called neurons. These neurons can accept inputs, apply weighting coefficients and feed their output to other neurons that continue the process through the network to the eventual output. Some neurons may send feedback to earlier neurons in the network. Neural networks are trained to deliver the desired result by an iterative process where the weights applied to each input at each neuron are adjusted to optimize the desired output.

Questioning how the MLP network compares as a classification method, we then analyze the data using a traditional statistical model, called the binary logistic regression. This method is a variant of linear regression which is the most useful when modelling the event probability for a dependent variable with two outcomes (i.e. acceptance or rejection in this context). Since the probability of an event must lies between 0 and 1, it is impractical to model probabilities with a linear regression method which allows the

dependent variable to take values larger than 1 or smaller than 0. However, the binary logistic regression method achieves this by linking the range of real numbers to the range between 0 and 1.

3 Experimental results

Table 1 shows the case information and summarizes the classification results produced by the MLP network and the binary logistic regression. For both models, we implement 10 runs on each data instance from the Toronto dataset. We employ the notation of (Qu et al. 2008). At each run, we use a random sample of 70% cases to create the prediction model, setting the remaining case to validate the analysis. Columns 4-7 and columns 8-11 list the statistical results on the percentage of cases that are correctly classified, for the MLP network and the logical regression method respectively.

Table 1 Classification results on 11 timetabling problems

Data	Case info		MLP neural network				Binary logistic regression			
	|var|[+]	|spl|[*]	Max %	Min %	Mean %	Dev.	Max %	Min %	Mean %	Dev.
car91	228	2654	60.1	55.9	57.7	1.5	60.8	54.6	57.8	1.7
car92	182	2794	61.5	58.9	60.4	0.7	62.5	59.3	60.7	1.0
ear83	191	3210	68.2	63.3	65.4	1.6	66.7	62.9	64.7	1.4
hec92	82	199	76.0	52.2	65.6	6.9	54.7	45.6	48.9	2.9
kfu93	231	2696	75.9	70.4	73.6	1.7	78.4	72.9	75.0	1.7
lse91	191	4859	59.0	54.5	56.0	1.2	59.4	47.1	52.7	4.6
sta83	140	6695	98.3	95.6	97.1	0.9	98.6	97.9	98.1	0.2
tre92	131	4084	75.3	67.9	71.3	2.4	72.9	69.0	71.0	1.2
uta93	208	3939	68.2	63.3	65.6	1.5	65.8	60.0	63.4	1.9
ute92	185	3355	64.2	60.0	62.8	1.3	70.4	49.4	55.8	8.4
yor83	182	264	72.0	57.5	64.8	4.9	73.2	51.3	60.8	6.2
Ave.	**177**	**3159**	**70.8%**	**63.6%**	**67.3%**	**2.2**	**69.4%**	**60.9%**	**64.4%**	**2.8**

[+] "|var|" denotes the number of independent variables, i.e. the number of exams.
[+] "|spl|" denotes the number of samples or cases.
"%" denotes the percentage of the cases that are correctly classified.

The results in Table 1 demonstrate the existence of some global patterns that are hidden in the solutions defined by the hyper-heuristic search space. On average, the MLP neural network performs slightly better than the binary logical regression method, with a mean correct classification rate of 67.3% (versus 64.4% for the regression model). Considering the number of dependent variables (177 on average) and the size of the sample set (3159 on average), the classification rate should be fluctuated around 50% for all instances and all runs if there is no pattern hidden in the data. Of course, one might argue on our above supposition because for some instances (such as car91 and lse91), the average classification rates are dissatisfactory (less than 60%). However, this only means that the global patterns are hard to detect in the given samples of those instances.

5 Conclusions and future research

The major contribution of our work is that, as far as we are aware, this is the first attempt to apply data mining techniques to design a "stand alone" component that can be incorporated into any constructive hyper-heuristics as a post-processor. The processor is able to recognize the patterns hidden in the resulting solutions, so that the hyper-heuristics may implement more iterations of search while keeping the CPU time not significantly increased or even slightly reduced. We notice that there is another paper in the literature (Thabtah and Cowling, 2008) which also falls into the domain of data mining. However, the methodology proposed in that paper is essentially no different (in this context) from the others because it only uses associative classification as a higher level method to choose between lower level heuristics, and thus it could not reduce any computational complexity that a hyper-heuristic causes.

Our work opens a wide area for further research. Apart from the study of more advanced classification techniques such as evolutionary neural networks, this work sheds a light on the development of more intelligent knowledge-based decision support system. Also, our approach in its current form is only

proposed for the constructive hyper-heuristics, we are looking at its applications on the improvement hyper-heuristics.

In addition, with the aid of a "pattern recognition" processor, we envisage such a knowledge-based system in the near future: once a hyper-heuristic has accumulated a certain amount of data worthy mining, this processor would be invoked automatically to work as a filter. The lifecycle of the filter and percentage of data entering the filter could be controlled by a number of parameters representing our own understanding and confidence. These parameters, of course, can also be adjusted adaptively depending on the changing environment. With the hyper-heuristic search in progress, the "pattern recognition" processor could be prohibited, revitalized and updated. Hence, the hyper-heuristics can be truly knowledge-based and significantly speeded up, allowing more heuristics recruited at the lower level and more iterations of search operated at the higher level.

Acknowledgements

The work was funded by the UK's Engineering and Physical Sciences Research Council (EPSRC), under grant EP/D061571/1.

References

Aickelin U, Burke EK, Li J (2007) An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. Journal of the Operational Research Society 58: 1574-1585.

Bozdogan H (2004) Statistical data mining and knowledge discovery. CRC Press.

Burke EK, Kendall G, Newall J et al (2003) Hyper-heuristics: an emerging direction in modern search technology. in: Glover F, Kochenberger G (eds) Handbook of meta-heuristics, 457-474. Kluwer.

Burke EK, Li J, Qu R (2008) A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. European Journal of Operational Research (accepted).

Burke EK, McCollum B, Meisel A, Petrovic S, Qu R (2007) A graph-Based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176: 177-192.

Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2008) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling. DOI: 10.1007/s10951-008-0060-1.

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation (1986). in: Rumelhart D.E., McClelland JL (eds) Parallel distributed processing: explorations in the microstructure of cognition, 318-362. Cambridge, MA: The MIT Press.

Thabtah F, Cowling P (2008) Mining the data from a hyper-heuristic approach using associative classification. Expert systems with applications 34: 1093–1101.