Distributed Choice Function Hyperheuristics for Timetabling and Scheduling

Andy Gaw, Prapa Rattadilok, and Raymond S K Kwan

School of Computing
University of Leeds
Leeds, LS2 9JT, UK
{gaw|prapa|rsk} @comp.leeds.ac.uk

Hyperheuristics are a powerful emerging search technology [1]. Search algorithms are often constructed from a collection of simple neighbourhood moves referred to as low level heuristics. Rather than hard-wiring such simple moves, hyperheuristics employ a domain independent driver that iteratively makes dynamic decisions on which simple move(s) should be executed next. The selected heuristics can be knowledge-poor heuristics like simple add, drop and swap moves or complete algorithms more akin to Meta-heuristics. Cowling et al. [2, 3], and Soubeiga [4] proposed a Choice Function based hyperheuristic driver, which has components designed for search intensification and diversification and incorporates some simple learning capability. Selection is made based on three factors: the previous performance of the heuristic, the performance of the last heuristic pair and the time the heuristic was last used.

This paper reports on on-going research in the design of choice function hyperheuristics, the modelling of general-purpose low level heuristics, and the exploitation of parallel computing platforms for hyperheuristics.

A prototype system was built to for the PATAT timetabling competition problem (see http://www.idsia.ch/Files/ttcomp2002), as a case study and trial of the basic design. The research indicates several areas in which performance can be improved.

A key aim of the research is to simplify the development of effective low-level heuristics, and to permit their reuse in different problem applications. An abstract representation of a working solution is used, which includes meta-data on the state and history of the solution, and the progress of the search. A problem definition, including the objective function components, input and output routines, and the details needed to specify a solution, must be written for each new problem type. This, in effect, translates the abstract solution representation into an explicit solution to a particular problem.

Solution fitness (quality) is evaluated using a weighted sum of terms, with terms determining feasibility having far higher weights than those which determine only the degree of optimality. Heuristic performance is then based on change in fitness. This causes problems at the crossover between feasible and infeasible solutions if a single choice function is used for both phases, because of the change in magnitude of the performance scores. We therefore propose the use of separate fitness functions for feasible and infeasible solutions.

Heuristics may be designed or used for different purposes. They are often described as intensifying or diversifying (respectively seeking to explore nearby, similar solutions, or move to other, different ones). We consider a third type, the conditioning heuristic, which aims not to improve the solution directly, but to create a state from which others can then generate an overall improvement. If the aims are different, then the performance metrics should reflect this, and we propose the use of separate performance and selection functions for groups of heuristics with different aims. These changes naturally lead to the idea of a multi-level hyperheuristic, with a top level selection of a hyperheuristic based on solution feasibility and strategy, followed by the selection of a suitable low-level heuristic for the task. This also makes the decision-making processes in the search more explicit.

The evaluation of successive pairs of low-level heuristics in the choice function implies that conditioning effects occur, but the original method accounts neither for the influence of heuristics other than the immediately preceding one, nor for the fact that the preceding heuristic may not have changed the current solution in any way. We propose a more flexible system of evaluating this type of influence.

It is essential that the system makes full use of whatever information is available. The original choice function uses the change in fitness in the solution caused by a heuristic and the time taken to achieve it. We record more detailed data, including the contribution to fitness of individual assignments in the solution and individual terms in the fitness function, and data on the times at which changes were made. This is exploited by both the low-level heuristics and hyperheuristics, making them more effective. Hyperheuristics can locate the individual assignments in a solution which are responsible for poor solution fitness, and, by ranking heuristic performance against each individual component of the fitness function, can select the most appropriate heuristics based on the current state of the solution. Low-level heuristics can be designed to target specific areas for improvement, using the meta-data to target the moves most likely to give improvement. Diversification heuristics may target those assignments which have seen relatively little change, whereas conditioning heuristics may aim to make available the most flexible (and therefore most changed) resources.

Parallel computer architectures open up many possible variations of the hyperheuristic scheme, each taking advantage of the simultaneous use of numerous processors. In particular, emerging and rapidly advancing Grid technologies may one day provide cheap access to a near unlimited number of distributed processors for use by the hyperheuristic. Two variations of distributed hyperheuristic are being investigated. In the first, overall control of the search is managed by a single hyperheuristic driver running on one of the processors. In each iteration, the available processors are partitioned amongst the low level heuristics to be executed. Each processor executes only one low level heuristic, but many processors may be executing the same low level heuristic on different parts of the timetable. The strategy used depends on the historical performance of the low level heuristics and a balance between the need for exploration and exploitation of the search space. The second variation consists of multiple instances of controlling hyperheuristics, each with their own subordinate processors. The controllers collaborate as agents in parallel, and exchange information and good solutions on a regular basis.

The traditional flaws of the centralised Master/Slave approach (e.g. Poor scalability, over-reliance on a small subset of nodes, single point of failure) are partially overcome by replicating Masters in several locations, but ultimately we feel a less hierarchical, peer to peer approach may be more efficient and robust.

References

1. Burke, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In Glover, F., Kochenberger, G., eds.: Handbook of Meta-Heuristics. Kluwer (2003). 457–474
2. Cowling, P., Kendall, G., Soubeiga, E.: A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In: 4th Metaheuristics International Conference, MIC 2001. Extended abstract. (2001). 127–131
3. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: a Tool for Rapid Prototyping in Scheduling and Optimisation. In: Applications of Evolutionary Computing. EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIME/EvoPLAN. Proceedings. Springer-Verlag, Berlin, Germany (2002). 1–10
4. Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling. PhD thesis, School of Computer Science and Information Technology (2003).