Dynamically Configured λ-opt Heuristics for Bus Scheduling

Prapa Rattadilok and Raymond S K Kwan

School of Computing, University of Leeds, UK
{prapa, rsk}@comp.leeds.ac.uk

Bus scheduling is a complex combinatorial optimization problem [3], [10]. The operations planning and scheduling process starts off with the designing of a timetable of trips that have to be served by buses. Each trip has a starting time/location and a destination time/location.

Bus scheduling involves assigning a set of trips to a set of buses such that:

1. The sequence of the trips for each bus is time feasible: no trip precedes an earlier trip in the sequence
2. Each trip is served by exactly one bus

There are two types of operational cost involved, these are layover (idle time) and dead running (relocating the bus between locations without passenger, this includes leaving and returning to the depot). The objective is to minimise the operational cost and the number of buses. Given the nature of the problem it is almost always impossible to reduce the layover and dead running cost to zero. The complexity of the problem quickly increases as the number of the depots and the types of vehicle increases.

A solution is said to be λ-optimal (or simply λ-opt) if it is impossible to obtain a better solution by replacing any λ relation instances by any other set of λ relation instances. The λ-opt heuristic [5] is based on this concept of λ-optimality where in each trial, λ instances of the chosen relation (mapping between problem components, e.g. bus trips to bus's timeslots) in the working solution are exchanged. The trial process continues until a move that satisfies the specified acceptance criteria is found. The accepted move is then used to update the working solution. Computational time rapidly increases for increasing values of λ, as a result, the values $\lambda = 2$ and $\lambda = 3$ are the most commonly used. When λ = number of relation instances we have an exhaustive search where every possibility is tried. In many applications $\lambda = 2$ is powerful enough to yield near optimal solutions in a fraction of the time needed for an exhaustive search.

E. K. Burke, H. Rudová (Eds.): PATAT 2006, pp. 473–477. ISBN 80-210-3726-1.

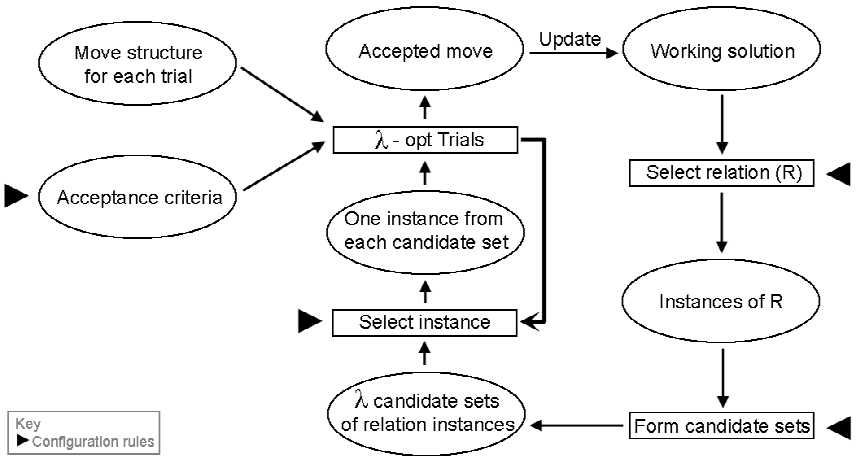


Fig. 1. λ-opt Heuristic

Figure 1 illustrates the general scheme of a λ-opt heuristic. Apart from specifying value of λ, there are four other decisions that need to be taken to fully configure a λ-opt heuristic. The relative merits of the options are not obvious for example, which relation should be selected, which trial solution should be accepted, etc. Different combinations of design options constitute different search strategies. If dynamically configured, these design options could adapt the heuristics to suit the current state of the search process, but choosing "the right" combination of design options is not a trivial task. Figure 2 illustrates the hyper-heuristic framework for the λ-opt heuristic dynamic configuration.

BOOST [4] applies an object-oriented paradigm to solving the bus scheduling problem. Object classes like 'Vehicle activities' (represent bus links, the connection between two bus trips) and 'Link swap rules' are used. The link swap rules or the scheduling heuristics provide the search moves for the optimization. For example, swapping valid bus links (bus links that correctly follow the sequence) with invalid bus links or valid bus links with start activities links (in order to find earlier starting time). These link swap rules are the extended functions that made up the algorithm in VAMPIRES [8]. These heuristics are the 2-opt scheduling heuristics. Although the major benefit of applying object-oriented models was to increase the extendibility of the application, the heuristics used inside the link swap rules class still require much of the problem specific knowledge. Configuring the 2-opt heuristic use within the system is still a tedious task. The heuristics were hard-wired and the design options were specified by the system designer. A more desirable approach would be to have the design options to be configured by the system itself based on the current search situation.

Hyper-heuristics are a powerful emerging search technology [1]. Search algorithms can be constructed from a collection of simple neighbourhood moves referred to as

low-level heuristics. Rather than hard-wiring such simple moves, hyper-heuristics employ a domain independent driver that iteratively makes dynamic decisions on which simple move or moves 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.

Soubeiga [9] proposed a choice function based hyper-heuristic driver, which has components designed for search intensification and diversification and incorporates some simple learning capability. The choice function provides the ranking of the low-level heuristics, based on information about the individual performance of each low-level heuristic, joint performance of pairs of heuristics, and the amount of time elapsed since the low-level heuristic was last called. The low-level heuristic performance is calculated in terms of the amount of solution improvement the low-level heuristic has achieved and the time it used to obtain this improvement. This choice function has been applied to select problem specific low-level heuristics on several timetabling and scheduling problems [2], [6].

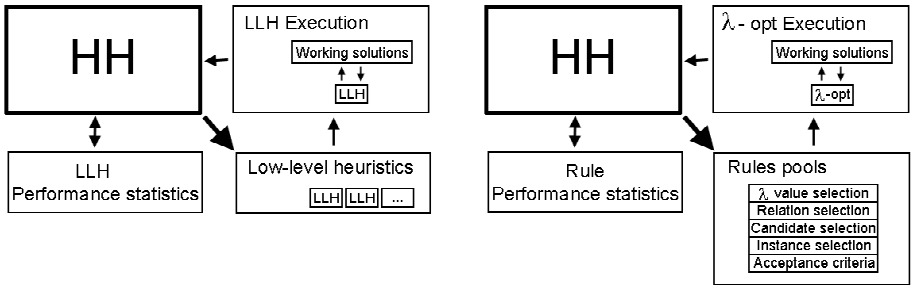
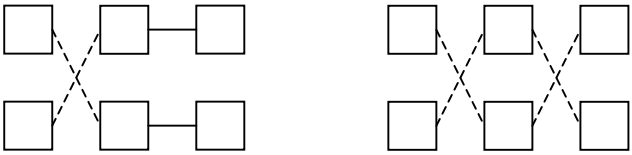


Fig. 2. The Hyper-heuristic framework for the static heuristics (left) and λ-opt heuristic dynamic configuration (right)

There are many possible useful relations for bus scheduling, for example, a relation of multiple bus trips to multiple bus timeslots of the same bus, a relation of a single bus trip to a single bus's timeslot. In the initial experiment, only the simple type of relation is experimented with, a trip to a bus's timeslot. Instead of trying to swap exhaustively between the 2 candidate sets ($\lambda = 2$), the trial swap performs the exchange between a selected candidate instance (the first candidate) and all other candidate instances (the second candidate). The first candidate is obtained by selecting randomly from the top ten instance of the selected ordered candidate set. Three kinds of instance order are used, based on four different kinds of cost (dead run, idle time, start activity and end activity), max-min, min-max and random. After all candidates are tried, the best pair will be selected and the update will be made to the schedule. Figure 3 illustrates the move structures used. Each square represents each bus trip and the dash line represent the bus link where the crossover is made.



(a) Simple crossover (b) Multiple crossover

Fig. 3. Move structures

Similar rules were previously used on a university timetabling problem [7]. A few minor modifications were needed to make it suitable to bus scheduling, for example, the constraint violation in timetabling is considered as the operational cost in bus scheduling. The starting solution is generated using a simple greedy assignment algorithm. The trips are assigned to the bus based on their starting time (which was given in the problem data set). A new bus is started once no more trips can be assigned to the existing bus. The process continues until all buses are tried. If there are any more trips left un-assigned then the rest of the trips are assigned to any empty bus timeslot.

In the initial experiment, all rules are expanded into every possible configuration (every possible low-level heuristic that could be obtained within the provided set of rules). The aim of this initial experiment was to investigate the feasibility of this general framework. At each iteration, the hyper-heuristic selects an expanded combination based on their score (as if it was to select a low-level heuristic).

The initial results obtained on a test data set is comparable to a highly specialised approach [4]. The fact that similar or identical low-level heuristics can be reused demonstrates the main strength of hyper-heuristics, flexibility. Further tuning and optimisation can improve performance but the fact that that isn't needed can be seen as a major success of the system. One possible fine-tuning is to supply hyper-heuristic with a more problem specific rules. More details results will be presented at the conference.

The initial investigation has revealed possible obstacles that need to be overcome before this approach can be considered completely successful. For example, the hyper-heuristic is performing less well as the number of rules increases. Rather than selecting an expanded configuration the hyper-heuristic has to be configured more dynamically. It is believed that this dynamic configuration is too big a task for a single choice function alone. Our current investigation is looking into a hierarchical hyper-heuristic where each layer makes different decisions but all leading to one goal of dynamically selecting the suitable configuration.

References

1. Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S.: Hyper-Heuristics (2003): "An Emerging Direction in Modern Search Technology". In: Glover, Fred, and Kochenberger, Gary A. (eds.): *Handbook of Meta-Heuristics*. Kluwer Academic Publishers, Boston, USA, 457-474.
2. Cowling, P., Kendall, G., Soubeiga, E. (2002): Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In: Cagnoni, C. et all (eds.): *EvoWorkShops. Lecture Notes in Computer Science*, Vol. 2279. Springer-Verlag, Berlin Heidelberg (2002)
3. Daduna, J. R. and Paixao, J. M. P. (1995): "Vehicle scheduling for public mass transit – An Overview". In: Daduna, J. R., Branco, I., Paixao, J. M. P. (eds.): Computer-aided transit scheduling, Springer-Verlag 76-90
4. Kwan, R. S. K. and Rahim, M. A. (1999): "Object Oriented Bus Vehicle Scheduling – the BOOST System". In Wilson N. H. M. (ed.), Computer-aided transit scheduling, Springer, Berlin, pp-177-191, 1999.
5. Lin, S. and Kernighan, B. W. (1973): "An Effective Heuristic Algorithm for the Travelling Salesman Problem". *Operations Research* Vol. 21, 498-516.
6. Rattadilok, Prapa, Gaw, Andy, Kwan, R. S. K. (2005): "Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling". In Burke, E. and Trick, M. (eds.): *Practice and Theory of Automated Timetabling V*. Springer LNCS. Vol. 3616/2005 pp. 51-67
7. Rattadilok, P. and Kwan, R. S. K. (2005): "Hyper-heuristic Driven Dynamically Configured λ-opt heuristics". In: Proceeding of Metaheuristic International Conference 2005
8. Smith, B. M. and Wren, A. (1981): "VAMPIRES and TASC: two successfully applied bus scheduling programs". In: Wren, A. (ed.): Computer scheduling of public transport. North Holland 97-124
9. Soubeiga, Eric (2003): "Development and Application of Hyperheuristics to Personnel Scheduling". *Ph.D. Thesis*, University of Nottingham.
10. Wren, A. (1981): "General review of the use of computers in scheduling buses and their crews". In: Wren, A. (ed.): Computer scheduling of public transport. North Holland, 3-16