Memory Length in Hyper-heuristics: An Empirical Study

Ruibin Bai, Edmund K. Burke, Graham Kendall
School of Computer Science & IT
University of Nottingham
Nottingham NG8 1BB, UK
Email: rzb|ekb|gxk@cs.nott.ac.uk

Barry McCollum
Department of Computer Science
Queen's University Belfast
Belfast, BT7 1NN, UK
Email: b.mccollum@qub.ac.uk

***Abstract*— Hyper-heuristics are an emergent optimisation methodology which aims to give a higher level of flexibility and domain-independence than is currently possible. Hyper-heuristics are able to adapt to the different problems or problem instances by dynamically choosing between heuristics during the search. This paper is concerned with the issues of memory length on the performance of hyper-heuristics. We focus on a recently proposed simulated annealing hyper-heuristic and choose a set of hard university course timetabling problems as the test bed for this empirical study. The experimental results show that the memory length can affect the performance of hyper-heuristics and a good choice of memory length is able to improve solution quality. Finally, two dynamic approaches are investigated and one of the approaches is shown to be able to produce promising results without introducing extra sensitive algorithmic parameters.**

I. Introduction

Considerable research has been carried out in developing optimisation methodologies to tackle various challenging problems [1]. Despite many successful applications being reported in the literature, using these techniques normally requires a high level of expertise and experience. In many cases, an implementation of a particular technique (or choosing the best one from a group of those available) involves significant parameter tuning. Hyper-heuristics are an emerging search technique which aims to raise algorithms' domain independence level such that minimal expertise is needed in order to apply it to solve a given problem. Although many hyper-heuristic frameworks have been proposed and tested on various scheduling and application problems recently [2]–[11], very limited research has been carried out with regard to some of important memory issues in hyper-heuristics. In this paper, we focus on the issues surrounding how different memory lengths affect the performance of hyper-heuristics and what type of self-adaptive approaches can be used to dynamically change the memory length at different stages of the search. The paper is structured as follows: section II gives a brief review of hyper-heuristics and issues that are worthy of further investigation. In section III, a university course timetabling problem is introduced and chosen as a test problem for this study. Section IV describes the heuristic selection function and weight adaption strategies that are used in the hyper-heuristic algorithm investigated in this paper. Two dynamic memory length approaches are proposed and investigated in section V and section VI concludes the paper.

II. Hyper-heuristics

Hyper-heuristics are high-level heuristic strategies that choose between heuristics to solve a problem given in hand [12], [13]. Hyper-heuristics search in the heuristic space as opposed to most of meta-heuristics which operate on the solution space. One of the ideas behind the hyper-heuristics is to manage a group of simple, low-level knowledge of heuristics in such a way that the algorithm can adaptively combine these simple heuristics to tackle the problems under consideration, with the help of some classifier systems or machine learning techniques to adapt the search to the problem domain.

Hyper-heuristics differ from widely adopted multi-neighbourhood meta-heuristic approaches in terms of its domain independence and adaptive control of heuristic switching based on adaptive learning techniques. For example, one of the most popular classes of multi-neighbourhood approaches is variable neighbourhood search (VNS) which systematically switches neighbourhoods in a predefined sequence so that the search can explore increasingly distant neighbourhoods of the current solution [14]. Here, designing a set of neighbourhood structures and arranging them into an increased cardinality sequence requires a good understanding and knowledge of both the problem domain and the search algorithm. Some other applications of multi-neighbourhood approaches allow users to change the sequence of the neighbourhood during the search either randomly or based on a predefined rule, for example, [15]–[17]. However, the neighbourhood structures of many of these approaches are very complicated and rely heavily on problem domain structures. Designing of these neighbourhoods requires a high level of expertise and experience. In contrast, hyper-heuristics aim to provide a system which can considerably reduce the requirement of users knowledge over the problem under consideration by allowing the algorithm to adapt itself to the different search scenarios either online or offline.

Apart from designing sophisticated neighbourhood moves, another central issue for multi-neighbourhood approaches is to determine a strategy of systematically changing between neighbourhoods during the search or assign a probability distribution with which each neighbourhood is chosen. Currently, the decision of such strategies is still an art, rather than a science, and often involves significant experiments and parameter tuning. Although those finely tuned algorithms are able

to produce high quality results for the problems (or problem instances) they are designed for, the performance of these problem-tailored algorithms may be decreased (sometimes drastically) if some conditions or properties of the problem change. One may have to tune the parameters again or redesign the algorithm completely. This may be appropriate for some applications where ample re-development time is allowed and a high quality solution is a priority. However, there are many other scenarios where users are only interested with satisfactory solutions and the problem situations change very quickly so that time-consuming parameter tuning is not allowed. In this case, a flexible, adaptive system with satisfactory solution quality is preferable to a highly sophisticated, but "brittle" system.

Several hyper-heuristic approaches have been proposed and applied to difficult scheduling problems. The heuristic selection mechanisms that were used in these approaches are either based on offline classifier models [5], [8]–[10] or online machine learning techniques [2], [4], [6], [7]. Previous research has shown that hyper-heuristics with online learning have advantages over offline learning hyper-heuristic approaches in terms of solution quality and algorithm domain independence [6]. However, offline learning hyper-heuristics are generally computationally faster since no online learning is involved.

Several online learning hyper-heuristic mechanisms have been based on the ideas of reinforcement learning [18], in which historical information is utilised in order to choose heuristics more intelligently in the next stage of the search. That is, each heuristic is subject to a reinforcement procedure to increase or reduce its chances of being chosen in future. To do this, some quantitative measurements are generally used to valuate the performance of each heuristic throughout the search. Preferences or priorities are then applied to those heuristics that have obtained good overall performance during the search history. However, few studies have been carried out to investigate the issues of what types of information to be exploited and in which way. Currently, most hyper-heuristic approaches have used long-term memories. That is, information gathered during the early stage of the search has the same influence as the information which was obtained only recently. For example, in [4], [2] and [7], the utility weights of heuristics are based on the entire historical information with equal impact, which may be not appropriate. It is also possible that these values are getting either very large or very small. To constrain these values, within a meaningful range, all the three papers employed a lower bound and a upper bound for these values. However, this method introduces two new parameters into the algorithm which may harm the generality of the method. In addition, as pointed out in [7], improving moves during the middle and later stages of the search are expected to be low, which leads to the situation that all the heuristic utility weights are approaching their lower bounds and the hyper-heuristic starts to operate in a random way.

In this study, we investigate these fundamental issues in the framework of a simulated annealing hyper-heuristic that was studied in [19]. The reason we choose this simulated annealing hyper-heuristic algorithm is that it has demonstrated significant improvement over other types of hyper-heuristics on several difficult scheduling and optimisation problems without lowering the generality of the algorithm [19]. See Fig. 1 for a diagrammatic presentation of the algorithm. The main features of this simulated annealing hyper-heuristic include a stochastic heuristic selection mechanism, a simulated annealing acceptance criterion and a short-term memory [19]. The heuristic selection mechanism plays a pivotal role. It constantly adapts the priorities of the low-level heuristics during the search. Initially, the heuristic selection mechanism does not know which heuristic will perform better than any other. Therefore, the heuristics are selected uniformly. As the search proceeds, the heuristic selection mechanism starts to apply preferences among different low-level heuristics by learning from, and adapting to, their historical performance. Therefore, the heuristics that have been performing well are more likely to be chosen. To successfully apply a selected heuristic, the simulated annealing acceptance criterion also has to be satisfied. That is, once a decision is made by the heuristic selection mechanism, the chosen heuristic is then applied to the current solution. The simulated annealing acceptance criterion is employed to decide whether to accept this heuristic move or not. Information about the acceptance decisions by the acceptance criterion is then fed back to the heuristic selection mechanism in order to make better decisions in future.

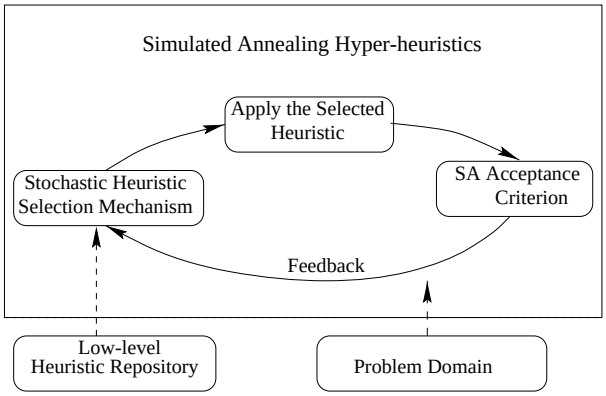

Fig. 1. A diagram of the simulated annealing hyper-heuristic

More specifically, in this paper we are interested in investigating how different memory lengths affect the performance of the hyper-heuristics and how the algorithm can automatically tune the influence of the information attained at different stages of the search.

A typical university course timetabling problem is chosen as a test bed to investigate these fundamental issues.

III. Problem Domain: Course Timetabling

The university course timetabling problem involves assigning a given number of events (including lectures, seminars, labs, tutorials, etc) into a limited number of time-slots and rooms subject to a given set of constraints. Several problem

models have been introduced in the literature due to the different practical requirements by different universities [20]. In this paper, we consider a model that was originally presented in [21] and [22]. This model provides a representation of a typical university course timetabling problem and has been widely used in many academic publications. The model formulates the problem as follows:

Given a set of events E and a number of rooms R, with each room having one or more features out of feature set F. Each event is attended by a given number of students from student set S and requires some of the room features. The aim of the problem is to assign every event $e \in E$ to a time slot $t_k (k = 1, ..., 45)$ and a room $q \in R$ so that the following hard constraints are satisfied:

- No student should be assigned to more than one event in a time slot;
- The room assigned to an event should have sufficient capacity and all the features required by the given event;
- No more than two events can be scheduled in one room in a time slot.

The objective of the problem is to minimise the number of students involved in the following soft constraint violations (Scv):

- An event is scheduled in the last time slot of the day;
- A student has only one event in a day;
- A student has more than two consecutive events.

We adopted the same solution representation that was used in both [22] and [2]. In this representation, a solution was encoded as an E dimensional vector where a position in the vector denotes an event index and the value corresponds to the time slots assigned to the given events.

Instead of using many complex neighbourhoods as in many other research publications (for example [23], [24]), here we only use three simple heuristics:

H1 **Shift:** Move a random event from its current time slot to another random time slot.
H2 **Swap event:** Swap the time slots of two random events.
H3 **Swap timeslot:** Swap all events of two randomly selected time slots.

Similar to [22] and [2], room assignments are dealt with separately using a matching algorithm. All of the above heuristics were designed to ensure that the search proceeds in feasible space. If an infeasible solution is produced, the current solution is returned and the heuristic move is discarded.

A total of twenty problem instances are used here as testing instances, drawn from the International Timetabling Competition organised by the Metaheuristic Network [25]. The parameters of these instances are as follows: $\mid E \mid \in [350, 440]$, $\mid S \mid \in [200, 350]$, $\mid R \mid \in [10, 11]$, and $\mid F \mid \in [5, 10]$.

IV. Heuristic Selection Mechanism

A. Selection function

The heuristic selection mechanism in this simulated annealing hyper-heuristic is based on the ideas of stochastic ranking [26], in contrast with the deterministic approaches in most of the other hyper-heuristic algorithms. It has been shown that stochastic ranking is superior to other popular selection strategies in the context of an evolutionary algorithm [26]. It is also compatible with the stochastic nature of the simulated annealing. Here we use a simple selection method similar to roulette-wheel selection of a genetic algorithm. The probability that a heuristic i is selected from a collection of alternative low-level heuristics $\mathcal{H}$ is proportional to its weight w_i:

$$p_i = \frac{w_i}{\sum_{i \in \mathcal{H}} w_i} \tag{1}$$

B. Weight adaption

Weight adaption is the key phase for the success of the hyper-heuristics. This is also the area that is particularly focused in this paper. Both [4] and [2] used a reinforcement learning function with a long-term memory. The weight of heuristic i at an arbitrary iteration h can be calculated by:

$$w_{ih} = \sum_{k=1}^{h} r_{ik} \tag{2}$$

where r_{ik} is the positive (or negative) reinforcement value applied to heuristic i at iteration k. Therefore, the current weight of a heuristic is a collective reflection of its previous performance. At each iteration k, a positive reinforcement r_{ik} is rewarded to a heuristic that has improved the current solution in terms of objective value and a negative reinforcement is applied if it fails to do so. However, we observed that during the search, although some heuristics cannot improve the solution directly, they are still useful in creating some intermediate situations, from which the optimal solution (or a good quality solution) could be reached. It is not rational to penalise these heuristics. Hence in this research, we give a minor positive score to those heuristics which could transfer the state of the solution but could not improve the objective value. Meanwhile, we penalise those heuristics which could neither improve the current solution nor generate a new solution. The following reward/penalty values are used:

$$r_{ik} = \begin{cases} c, & \delta > 0 \\ -c, & \delta < 0 \\ \epsilon, & \delta = 0 \text{ and new solution} \\ -\epsilon, & \delta = 0 \text{ ant no new solution} \\ 0, & \text{if not selected} \end{cases} \tag{3}$$

where c is a positive value and ϵ is a relatively small positive value. In this application, we set $c = 1$, $\epsilon = 0.1$. As mentioned earlier, the majority of moves at the middle and latter stages of the search are non-improving, which would lead to most of heuristics being assigned a very large negative weight. Equation (3) does not generate probability distributions as we expected. Although the lower and upper bounds used in [4] and [2] can ensure that these weights are held within a reasonable range, setting these two bound parameters can be difficult. A bad choice of the parameters may lead the algorithm to becoming a random search since these weights are not able to

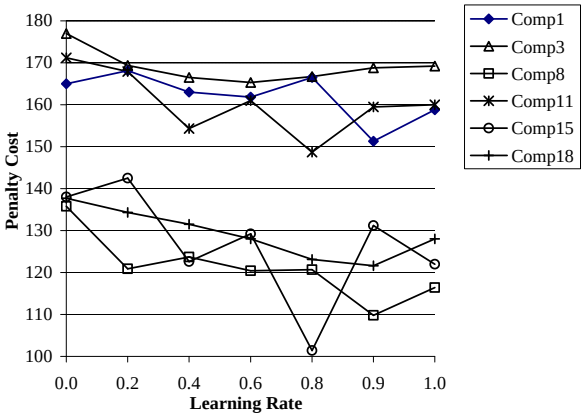

Fig. 2. The average performance of the simulated annealing hyper-heuristic with different learning rates over 10 runs for six randomly selected instances

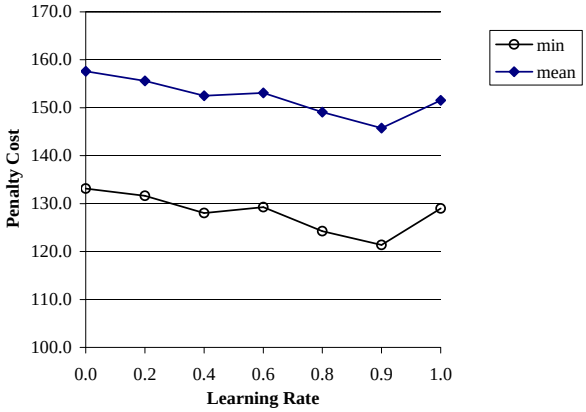

Fig. 3. The average results by the simulated annealing hyper-heuristic with different learning rates over all the 20 instances. min: The best objective in 10 runs. mean: The mean objective in 10 runs.

distinguish the performance between heuristics. In this paper, instead of assigning a lower and upper bounds for each weight, the following simple method is used to solve this problem. Let $w_{\min} = \min\{0, w_i \mid i \in \mathcal{H}\}$, the heuristic selection function is now defined as:

$$p_i = \frac{w_i + w_{\min}}{\sum_{i \in \mathcal{H}} (w_i + w_{\min})} \tag{4}$$

The following standard reinforcement learning function is used:

$$w_{ih} = \sum_{k=1}^{h} \alpha^k r_{ik} \tag{5}$$

where α is learning rate or discount factor to balance the influence of the rewards gained at different iterations during the search history. $\alpha = 0$ implies a random heuristic selection approach while $\alpha = 1$ corresponds to the long-term memory function used in [4] and [2]. Any α value between 0 and 1 would exponentially scale down the reinforcement applied to each heuristic over time. For example, for a learning rate of $\alpha = 0.5$, the influence of reinforcement will almost vanish after 10 iterations ($0.5^{10} \approx 0.001$).

Fig. 2 plots a comparison of the simulated annealing hyper-heuristic with different learning rates for six problem instances (due to the space reasons, only six randomly selected problem instances are presented here. Similar plots can be obtained for other problem instances). The parameters, with regards to simulated annealing, are set with the same values as in [19] except that the reheating strategy is turned off to eliminate potential factors which could have added disturbance to the experimental results. For each version of the algorithm, 10 independent runs were carried out and both the best results and average results were recorded. For each single run, the number of total iterations was set to K=2,000,000. It can be seen that in most cases, the algorithm without learning (i.e. $\alpha = 0$) performs worst when compared with the simulated annealing hyper-heuristics with different learning rates. It is also interesting to see that some of short-term memory based algorithms outperformed the algorithm with long-term memory or infinite memory as referred to in some other places (i.e. $\alpha = 1$). Unfortunately, it is difficult to determine an α value with which the simulated annealing hyper-heuristic performs best for every problem instance. For example, the best α value (among the values we have tried) for the instance Comp8 appears around 0.9. However, for the instance Comp15, the best α value is around 0.8. Indeed, this is a common problem associated with many optimisation techniques whose parameters are not only sensitive to the change of the problem domain, but also the change of instances from the same domain.

Fig. 3 presents the average results over 20 instances by the simulated annealing hyper-heuristic, with different learning rates. It can be seen that across 20 instances, the infinite memory ($\alpha = 1$) obtained better overall performance than the memoryless approach (i.e. $\alpha = 0$) but not as good as when the learning rate is set to 0.8 or 0.9. Over this set of problem instances, the hyper-heuristic with a learning rate $\alpha = 0.9$ performs best. However, one would have to run a considerable number of experiments to tune this parameter, which is not advocated in a hyper-heuristic approach. With hyper-heuristics, we expect the algorithms to be generic and self-adaptive in the sense that the parameters in the algorithm are either not sensitive to the changes of the problem or the parameters are dynamically tuned to the problem instance during the search. With this objective in mind, we investigated two alternative methods which are presented in the next section.

V. Dynamic Memory Length

From Fig. 2 it can be seen that the best α value changes from instance to instance. It might be beneficial to allow the algorithm to change the learning rate during the search. The first simple approach that we investigated (denoted by R) is similar to the mechanism used in the reactive GRASP (greedy randomised adaptive search procedure) [27]. The

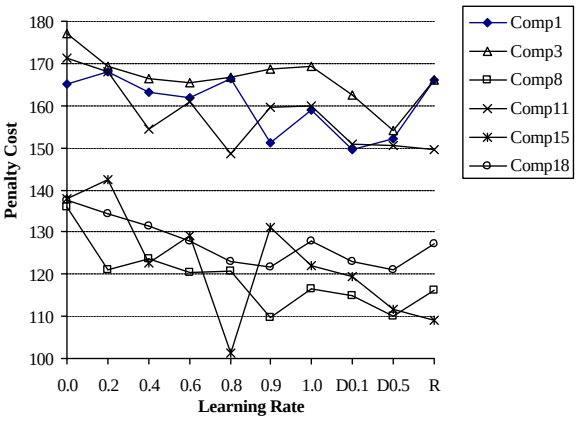

Fig. 4. A comparison of dynamic memory length approaches with static memory length approaches on six problem instances

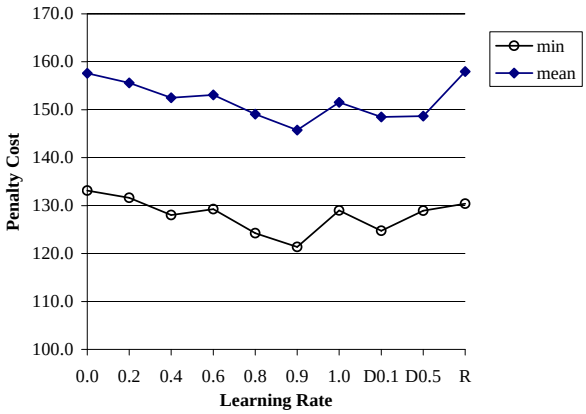

Fig. 5. The overall performance of the dynamic memory length approaches across all the 20 instances

search is divided into 10 identical sub-periods, with each period using a learning rate randomly selected from the set $\{0.5,0.6,0.7,0.8,0.9,0.99\}$.

The second approach is slightly more complex and is based on the following observation: at the middle and latter stages of the simulated annealing search, most of heuristic moves will be rejected and the ratio of improving solutions to non-improving solutions is generally small. In this scenario, it would need a longer period of historical information for the algorithm to distinguish between the performance of low-level heuristics. Therefore, it is reasonable to increase the memory length systematically as the search proceeds. With the learning function (5) used in this paper, this would be equivalent to increasing the learning rate α. In the paper, the search is, again, divided into 10 identical sub-periods. In the first sub-period, we assign a relatively small initial learning rate α_0. After each period, we increase the learning rate by the function $\alpha = \sqrt{\alpha}$, corresponding to roughly doubling the memory lengths each time. To test the sensitivity of the initial value α_0, both $\alpha_0 = 0.1$ and $\alpha_0 = 0.5$ were tried. For simplicity, these two versions of the algorithm are denoted by D0.1 and D0.5 respectively.

Fig. 4 and 5 present comparisons between these two approaches and the approaches with a fixed learning rates for six randomly selected individual instances and all the 20 instances respectively. From both figures it can be seen that the random learning rate approach (R) performed badly for most cases. However, the dynamic learning rate approaches D0.1 and D0.5 outperformed most of the static learning rate approaches. The results that were obtained by D0.1 and D0.5 are even comparable to the results by the best static learning rate approach (i.e. when $\alpha = 0.9$) which can only be achieved via a time-consuming parameter tuning process. Also note that the performances of D0.1 and D0.5 are very similar when compared with each other, which indicates that this dynamic memory length approach is not sensitive to the initial leaning rate α_0. The advantage of D0.1 and D0.5 over the static learning rate is that since there is no parameter tuning involved and the algorithm performed well across all 20 test instances.

Note that the purpose of this paper is to study the impact of the memory length on the performance of the hyper-heuristics. To exclude non-correlated disturbances, we turned off some strategies that are able to improve the algorithm's performance. Therefore, the results presented in this paper are not as good as those reported in [24], [28]. For example, we used significantly less iterations than in [24] which, if allowed, will significantly improve our results. Also, introducing a reheating strategy that is similar to [19] could also improve the results. In addition, we used three very simple heuristics as opposed to other algorithms that relied on much more complex heuristics/neighbourhood moves (for example, [23] and [24]).

VI. Conclusion

Memories are key components in the hyper-heuristics. This paper is concerned with the issue of memory length in hyper-heuristics. We specifically investigated how the memory length can affect the performance of a newly proposed simulated annealing hyper-heuristic. The empirical study on a university course timetabling problem showed that hyper-heuristics using a short-term memory produced better results than both the algorithm without memory and the algorithm with infinite memory. A possible explanation is that search landscape is different at different stages of the search and information about how a heuristic behaved during the past may only be useful for a limited period in the future. Information gathered a long time ago may be not valid any more. Unfortunately, the parameter of memory length seems to be sensitive to different problem instances and it is also time-consuming to tune this parameter to a given problem. In order to increase the level of domain independence of the hyper-heuristics and automatically set this parameter, a dynamic memory length approach has been investigated and proposed in this paper which has shown to be robust across different instances without a parameter tuning process.

In future, it would be worthy of studying other dynamic memory length approaches that can combine short-

term and long-term memories simultaneously in a hyper-heuristic framework. It would also be interesting to study the interactions between different heuristic selection functions and memory adaption approaches.

REFERENCES

[1] E. K. Burke and G. Kendall, Eds., *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Kluwer Academic Publishers, 2005.

[2] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[3] E. Hart, P. Ross, and J. A. Nelson, "Solving a real-world problem using an evolving heuristically driven schedule builder," *Evolutionary Computing*, vol. 6, no. 1, pp. 61–80, 1998.

[4] A. Nareyek, "Choosing search heuristics by non-stationary reinforcement learning," in *Metaheuristics: Computer Decision-Making*, M. G. C. Resende and J. P. de Sousa, Eds. Kluwer, 2003, ch. 9, pp. 523–544.

[5] E. K. Burke, S. Petrovic, and R. Qu, "Case based heuristic selection for timetabling problems," *Journal of Scheduling*, vol. 9, no. 2, pp. 99–113, 2006.

[6] T. Carchrae and J. C. Beck, "Applying machine learning to low-knowledge control of optimization algorithms," *Computational Intelligence*, vol. 21, no. 4, pp. 372–387, 2005.

[7] K. A. Dowsland, E. Soubeiga, and E. K. Burke, "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation," *European Journal of Operational Research*, vol. In Press, 2006.

[8] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995, pp. 1114–1120.

[9] P. Ross, J. G. Marin-Blazquez, S. Schulenburg, and E. Hart, "Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003*. Springer, 2003, pp. 1295–1306.

[10] W. Zhang and T. G. Dietterich, "High-performance job-shop scheduling with a time-delay td network," in *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. MIT Press, 1996, pp. 1024–1030.

[11] P. Rattadilok, A. Gaw, and R. S. K. Kwan, "Distributed choice function hyper-heuristics for timetabling and scheduling," in *The Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling*, ser. Lecture Notes in Computer Science Series, E. K. Burke and M. Trick, Eds. Springer, 2005, vol. 3616, pp. 51–70.

[12] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer, 2003, pp. 457–474.

[13] P. Ross, "Hyper-heuristics," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Springer, 2005, ch. 17, pp. 529–556.

[14] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[15] A. D. Gaspero and A. Schaerf, "A multi-neighbourhood local search with application to course timetabling," *Lecture Notes in Computer Science*, vol. 2740, pp. 262–275, 2003.

[16] R. K. Ahuja, J. B. Orlin, and D. Sharma, "A composite very large-scale neighborhoodstructure for the capacitated minimum spanning tree problem," *Operations Research Letters*, vol. 31, pp. 185–194, 2003.

[17] N. Musliu, A. Schaerf, and W. Slany, "Local search for shift design," *European Journal of Operational Research*, vol. 153, pp. 51–64, 2004.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[19] R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum, "A simulated annealing hyper-heuristic methodology for flexible decision support," *Submitted to Operations Research*, 2006.

[20] B. McCollum, "University timetabling: Bridging the gap between research and practice," in *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, PATAT[2006]*, 2006, pp. 15–35.

[21] O. Rossi-Doria, C. Blum, J. Knowles, M. Samples, K. Socha, and B. Paechter, "A local search for automated timetabling," in *Proceedings of the 4th International Conference on the Practice And Theory of Automated Timetabling [PATAT 2002]*, 2002, pp. 124–127.

[22] K. Socha, J. Knowles, and M. Samples, "A max-min ant system for the university course timetabling problem," in *Proceedings of the 3rd International Workshop on Ant Algorithm, ANTS 2002, Lecture Notes in Computer Science*, vol. 2463, 2002, pp. 1–13.

[23] S. Abdullah, E. K. Burke, and B. McCollum, "Using a randomised iterative improvement algorithm with composite neigjhbourhood structures for university course timetabling," in *The Proceedings of the 6th Metaheuristic International Conference [MIC05], Vienna, Austria, 22-26 Aug*, 2005.

[24] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria, "A effective hybrid algorithm for university course timetabling," *Journal of Scheduling*, vol. 9, pp. 403–432, 2006.

[25] Metaheuristic-Network, "International timetabling competition: Competition results," 2003, http://www.idsia.ch/Files/ttcomp2002/results.htm.

[26] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.

[27] M. Prais and C. C. Ribeiro, "Reactive grasp: an application to a matrix decomposition problem in tdma traffic assignment," *INFORMS Journal on Computing*, vol. 12, pp. 164–176, 2000.

[28] P. Kostuch, "The university course timetabling problem with a 3-phase approach," in *The Practice and Theory of Automated Timetabling V - Lecture Notes in Computer Science*, E. K. Burke and M. Trick, Eds., vol. 3616. Springer-Verlag, 2004, pp. 109–125.