4OR-Q J Oper Res (2012) 10:43–66
DOI 10.1007/s10288-011-0182-8

RESEARCH PAPER

A simulated annealing hyper-heuristic methodology for flexible decision support

Ruibin Bai · Jacek Blazewicz · Edmund K. Burke · Graham Kendall · Barry McCollum

Received: 11 November 2010 / Revised: 20 September 2011 / Published online: 23 November 2011

Abstract Most of the current search techniques represent approaches that are largely adapted for specific search problems. There are many real-world scenarios where the development of such bespoke systems is entirely appropriate. However, there are other situations where it would be beneficial to have methodologies which are generally applicable to more problems. One of our motivating goals for investigating hyper-heuristic methodologies is to provide a more general search framework that can be easily and automatically employed on a broader range of problems than is currently possible. In this paper, we investigate a simulated annealing hyper-heuristic methodology which operates on a search space of heuristics and which employs a stochastic heuristic selection strategy and a short-term memory. The generality and performance of the proposed algorithm is demonstrated over a large number of benchmark datasets drawn from two very different and difficult problems, namely; course timetabling and

R. Bai (✉)
Division of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK
e-mail: ruibin.bai@nottingham.edu.cn

J. Blazewicz
Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60-965 Poznan, Poland
e-mail: jblazewicz@cs.put.poznan.pl

E. K. Burke · G. Kendall
School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK
e-mail: ekb@cs.nott.ac.uk

G. Kendall
e-mail: gxk@cs.nott.ac.uk

B. McCollum
Department of Electronics, Electrical Engineering and Computer Science,
Queen's University Belfast, Belfast BT7 1NN, UK
e-mail: b.mccollum@qub.ac.uk

bin packing. The contribution of this paper is to present a method which can be readily (and automatically) applied to different problems whilst still being able to produce results on benchmark problems which are competitive with bespoke human designed tailor made algorithms for those problems.

Keywords Hyper-heuristics · Simulated annealing · Bin packing · Course timetabling

MSC classification (2000) 90-08: Computational methods

1 Introduction

Many real-world search problems represent particularly demanding research challenges. A wide range of methods and techniques (Glover and Kochenberger 2003; Burke and Kendall 2005) have been intensively investigated and studied to tackle such problems. However, many of these algorithms are either tailored for problem models by making use of problem-specific structures and properties, or they involve considerable parameter tuning. The performance of these algorithms often decreases (sometimes drastically) when some of the problem properties alter (even if only slightly). To improve the algorithmic performance for new problem instances, it is often necessary to invest considerable time and effort in tuning the parameters once again or even to completely redesign the algorithm. Moreover, these methodologies are selected and adapted by humans. We are concerned here with establishing scientific principles that are required to automate this process.

It should be recognised that problem modelling is a continual process. A model is only an approximation of reality. New observations or situations may lead to a refinement, modification, or replacement of a model because real-world problems often have to reflect an environment that is rapidly changing. Such observations motivate the development of flexible search techniques which can easily be adapted to respond to such changes.

Hyper-heuristics have recently received considerable research attention [see Ross (2005), Burke et al. (2009) for recent overviews of on hyper-heuristics] in order to address some of these issues. In Venkatraman and Yen (2005), a generic two-stage evolutionary algorithm framework was proposed for constrained optimisation problems. The algorithm avoids incorporating problem-specific characteristics but adaptively guides the exploration and exploitation using a non-deterministic ranking strategy. In this paper, we investigate and develop a simulated annealing hyper-heuristic framework which adopts a stochastic heuristic selection strategy (Runarsson and Yao 2000) and a short-term memory. We demonstrate the performance and generality of the algorithm over two very different and challenging optimisation problems: university course timetabling and bin packing.

2 Hyper-heuristics: an overview

One of the aims of hyper-heuristic research is to underpin the development of decision support systems which are applicable to a range of different problems and different

problem instances. Hyper-heuristics can be defined as "an automated methodology for selecting or generating heuristics to solve hard computational search problems" (Burke et al. 2009). This differs from most implementations of meta-heuristic methods which operate directly on a solution space. A hyper-heuristic methodology will explore a search space of heuristics. It is possible to make use of a repository of simple low-level heuristics or neighbourhood functions and to strategically change their preferences during the search in order to adapt to different situations and problem instances (Ross 2005; Burke et al. 2009). Note that hyper-heuristic research has been undertaken for a number of years although the term "hyper-heuristics" is relatively new. The roots of such work can be traced back to the 1960s (Fisher and Thompson 1963; Crowston et al. 1963) and throughout the 1980s and 1990s (Mockus 1989; Kitano 1990; Hart et al. 1998). This section gives a short overview of relevant hyper-heuristic methods. The interested reader can refer to Soubeiga (2003), Ross (2005), Burke et al. (2009)) for more detailed overviews.

It is possible to broadly categorise hyper-heuristics into those which represent *local search* methods and those which generate new heuristics. Constructive hyper-heuristics construct solutions from "scratch" by intelligently calling different heuristics at different stages in the construction process. Examples of constructive hyper-heuristic research can be seen in Fisher and Thompson (1963), Kitano (1990), Hart et al. (1998), Ross et al. (2003), Burke et al. (2006), Qu and Burke (2009). Local search hyper-heuristics start from a complete initial solution and repeatedly select appropriate heuristics to lead the search in promising new directions. This is the type of hyper-heuristic method with which this paper is concerned.

In local search hyper-heuristics, low-level heuristics usually correspond to several neighbourhood functions or neighbourhood exploration rules that could be used to alter the state of the current solution. Several types of local search hyper-heuristic have been investigated in the literature. Some hyper-heuristics draw upon ideas from reinforcement learning (Sutton and Barto 1998) to guide the choice of the heuristics during the search. Nareyek (2003) biased the heuristic selection probabilistically based on non-stationary reinforcement learning. Several weight adaptation strategies were tested and compared on two combinatorial optimisation problems.

Several evolutionary hyper-heuristics have also been investigated and studied. Kitano (1990) employed a genetic algorithm based hyper-heuristic to optimise neural network design. Instead of encoding the network configuration directly, his GA chromosome consisted of a set of rules that can be used to generate networks. This approach was shown to be superior to a conventional GA. Hart et al. (1998) solved a real-world scheduling problem using a GA based hyper-heuristic. The problem involved scheduling the collection and delivery of chickens from farms to processing factories. The GA was used to evolve a strategy to build a good solution instead of finding the solution directly. The experimental results showed this approach to be fast, robust and easy to implement. Other recent research work related to evolutionary hyper-heuristics includes Han and Kendall (2003), Ross et al. (2003), Terashima-Marin et al. (1999). Recently, Burke et al. (2010) used genetic programming as a hyper-heuristic approach to evolve new heuristics for the 2-dimensional strip packing problem. Computational tests show that the best heuristic evolved by the hyper-heuristic is competitive when compared against the best human-designed heuristics.

A tabu search based hyper-heuristic has also been developed which was effective on both a nurse rostering problem and a university course timetabling problem which demonstrated the level of generality of the method (Burke et al. 2003b). In this approach, the hyper-heuristic dynamically ranks a set of heuristics according to their performance in the search history. At each iteration, the hyper-heuristic applies the highest "non-tabu" heuristic to the current solution until the stopping criterion is met. Competitive results were obtained on both problems when compared with other state-of-the-art techniques. In Dowsland et al. (2006), their hyper-heuristic was enhanced within a simulated annealing framework and was used to solve a shipper sizes optimisation problem. The authors also discussed some heuristic performance measurement issues within this new hyper-heuristic framework.

Another type of hyper-heuristic has been proposed which uses a Bayesian heuristic approach to randomise and optimise the probability distribution of each heuristic call (Mockus 1989). This approach is based on the analysis of the average-case performance of the heuristics. It attempts to determine a set of parameters, or a probability distribution, so that the deviation from the global optimum is minimised. The method has been applied to a variety of discrete optimisation problems. See Mockus (1994, 1997, 2000) for further details.

Other search methods which have been employed as hyper-heuristics include graph based methods (Burke et al. 2007), choice functions (Cowling et al. 2001; Rattadilok et al. 2005) and case-based reasoning (Burke et al. 2006).

The simulated annealing hyper-heuristic we propose in this paper builds upon the methodologies presented in Bai and Kendall (2005) and Burke et al. (2003b).

3 A flexible simulated annealing hyper-heuristic (SAHH)

3.1 Framework of the proposed hyper-heuristic

In a similar way to other hyper-heuristic frameworks (e.g. Soubeiga 2003), our proposed simulated annealing hyper-heuristic has a *domain barrier* sitting between the hyper-heuristic and the problem domain. In order to facilitate a satisfactory level of *generality*, we restrict domain-dependent information from being transferred to the hyper-heuristic algorithm. However, non-domain information is allowed to pass through the barrier so that the hyper-heuristic can exploit this information. For example, the hyper-heuristic can be aware of the number of low-level heuristics available, changes in the evaluation function, whether a new solution has been generated or not and the distance between two solutions (i.e. how much two solutions differ), as this data is available no matter what problem domain we are operating on. Recall that the goal of this research is to be as "domain independent" as possible.

Our proposed hyper-heuristic is shown in Fig. 1 which is adapted from (Soubeiga 2003). It has the following features.

Firstly, it adopts a simulated annealing acceptance criterion (Lundy and Mees 1986) to alleviate the shortcomings of two simple acceptance criteria (improvement-only and accept all moves) that have been used in other hyper-heuristic approaches (Cowling et al. 2001; Nareyek 2003; Burke et al. 2003b). The simulated annealing acceptance

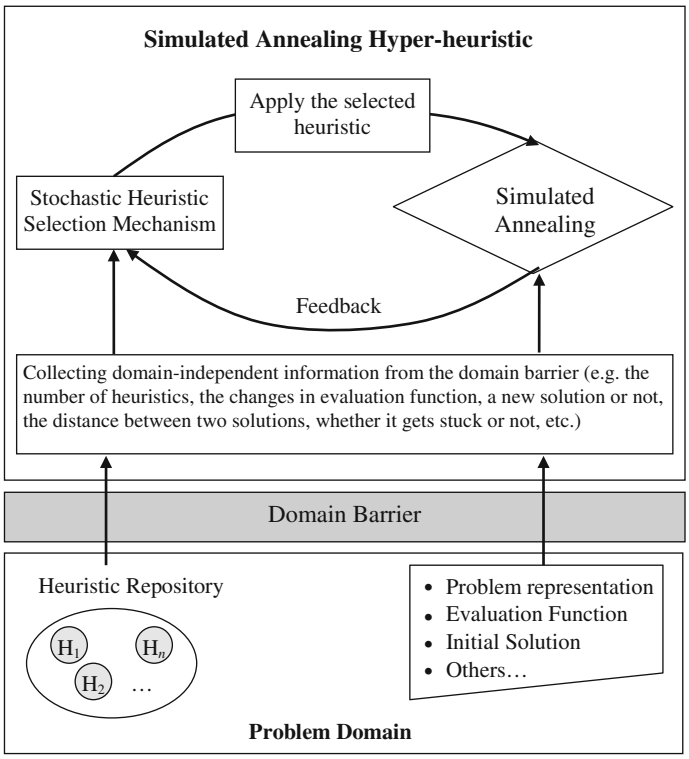

Fig. 1 The framework of our simulated annealing hyper-heuristic

criterion defines a probability, $p = \exp(-\delta/t)$, with which a given candidate solution is accepted.

Secondly, stochastic heuristic selection mechanisms are used instead of the widely used deterministic heuristic selection strategies. In Runarsson and Yao (2000), it was shown that stochastic ranking is superior to other popular selection strategies in the context of an evolutionary algorithm for constrained optimisation. The heuristic selection mechanism dynamically tunes the priorities of different heuristics during the search. Initially, of course, the heuristic selection mechanism does not know whether any heuristic will perform any better than any other. Therefore, all low-level heuristics are treated equally and the heuristic selection decisions are made randomly. While the search is proceeding, 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 the future.

```
procedure learn (LP)
begin
    c_i^total ++;
    Let δ be the difference in the evaluation function between s' (generated by heuristic H_i from s) and s;
    if (s' != s) c_i^new ++;  endif
    if (δ < 0)  c_i^accept ++; C_a ++; t_imp = t; f_r = false; endif
    if (δ > 0 && exp(-δ/t)>random(0,1)) c_i^accept ++; C_a ++;  endif
    if (mod(iter, LP) = 0)
        if (C_a/LP < r_e)
            f_r=true; t_imp = t_imp/(1- βt_imp); t= t_imp; s = s_best;
            for each i in {1,…,n} do
                if (c_i^total =0)  w_i = w_min;
                else w_i = max{w_min, c_i^new/ c_i^total}
                endif
                c_i^accept =0, c_i^new =0; c_i^total =0;
            done
        else
            for each i in {1,…,n} do
                if (c_i^total =0)  w_i = w_min;
                else w_i = max{w_min, c_i^accept/ c_i^total}
                endif
                c_i^accept =0, c_i^new =0; c_i^total =0;
            done
      endif
        C_a = 0;
    endif
end
```

Fig. 2 The learning procedure in the proposed simulated annealing hyper-heuristic

Thirdly, short-term memories are utilised, as opposed to long-term memories (as in some recent hyper-heuristic methods). The length of these memories (which we call a learning period) is much shorter compared with the whole search period. The underlying assumption is that each low-level heuristic may exhibit different levels of performance in different regions of the search space, or at different periods of the simulated annealing process. A heuristic that is effective in some regions of the search space might perform badly in other regions. If a heuristic frequently improves the current solution in the initial stages of the search, this does not necessarily mean that it will be effective in the middle or final phases of the annealing process. Therefore, we prefer to limit the memory of the hyper-heuristic.

In addition, the algorithm should distinguish between situations where a heuristic failed to generate a new solution and those where a heuristic returned a new solution but was unable to improve the objective function. Our hyper-heuristic encourages calls of the latter type of heuristic whilst reducing the first type, especially when the algorithm gets stuck at a local optimum.

In Fig. 3 we present the pseudo-code of our proposed algorithm which uses the learning procedure shown in Fig. 2. It assumes a minimisation problem but, of course, it is trivial to convert it to a maximisation problem. It can be seen that the main structure of the proposed algorithm in Fig. 3 is similar to a general simulated annealing algorithm. Note that in Fig. 3, a heuristic H_i can be considered as a mapping function from a current solution to a candidate solution. This candidate solution could be generated by a random sampling of a neighbourhood defined by H_i. Alternatively it could also

Initialisation:
Generate an initial solution s_0;
Define a set of heuristics H_i (i = 1,…, n), associate each heuristic H_i with three counters c_i^{accept} =0, c_i^{new} =0; c_i^{total} =0, a minimal weight $w_{\min}$ and an initial weight $w_i = w_{\min}$;
Set initial non-improving acceptance ratio r_s and stopping non-improving acceptance ratio r_e. Estimate the starting temperature t_s and stopping temperature t_e by r_s and r_e.
Set total iterations K, iterations at each temperature *nrep* and the length of single learning period LP;
Calculate the temperature reduction rate β=(t_s-t_e)·*nrep*/(K·t_s·t_e);
Set $t = t_s$; $t_{imp} = t_s$; *iter* = 0; C_a = 0; f_r=false;

Iterative improvement:
while (*iter* < K) **do**
  Select a heuristic (H_i) based on probability $p_i = w_i / \sum_{i=1}^{n} w_i$;
  Generate a candidate solution s' from the current solution s using heuristic H_i;
  Let δ be the difference in the evaluation function between s' and s;
  iter ++;
  if (δ <= 0 && new solution generated) $s := s'$; **endif**
  if (δ > 0 && (exp(-δ/t)>*random*(0,1)) $s = s'$; **endif**
  if (f_r = true) $t_{imp} = t_{imp}/(1- \beta t_{imp})$; $t = t_{imp}$;
  else if (mod(*iter*, *nrep*)=0) $t = t/(1+\beta t)$; **endif**
  endif
  call procedure learn(LP)
done

Fig. 3 Pseudo-code of the proposed simulated annealing hyper-heuristic (for a minimisation problem)

be a better solution obtained by evaluating partial or entire neighbourhoods. Therefore, the set of heuristics H can be a mixture of both types of heuristics, providing a balance between exploration and diversification. As might be expected, we found that, for our proposed SAHH, having more heuristics of the first type in the heuristic repository gives better performance in general for the problems we have tested (i.e. course timetabling and the bin packing). Coupled with simulated annealing criteria, the second type of low level heuristics may lead to occasional premature convergences during the search. In summary, our proposed algorithm has the following new features (see Figs. 2 and 3).

For each low-level heuristic, we associate a weight w_i ($w_{\min} \leq w_i \leq 1$) to represent its preference in comparison to the other heuristics. Initially, this weight is set to the minimal weight $w_{\min}$ (a very small positive value). The weights are then updated periodically.

- The starting temperature, t_s, and the stopping temperature, t_e, are estimated so that the initial and the final non-improving acceptance ratios (i.e. the ratio of the accepted non-improving moves to the total non-improving moves) approximately equal the predefined values r_s and r_e respectively. The temperature reduces according to Lundy and Mees' (1986) nonlinear function $t = t/(1 + \beta t)$ where $\beta = (t_s - t_e) \cdot nrep/K \cdot t_s \cdot t_e$ and *nrep* is the number of iterations at each temperature and K is the number of total iterations allowed.
- Since the performance of a heuristic may change at a different temperature or when in a different region of the search space, we measure a heuristic's performance based on the information gathered during a relatively short learning period, as opposed to the whole search history. Let LP ($LP < K$) be the length of a single learning period. Counters are used which track; c_i^{total} the total number of calls of heuristic i by the hyper-heuristic during the current learning period; c_i^{new} the total

number of new solutions generated by the heuristic i and c_i^{accept} counts how many of them have passed the simulated annealing acceptance criterion.

- A "reheating" strategy is also used and is triggered when the acceptance ratio is below the stopping acceptance ratio r_e. To do this, another counter (C_a) is used to record the total number of accepted heuristic calls during the current learning period. If the acceptance ratio is too low (i.e. $C_a/LP < r_e$), the system is switched to a "reheating" phase (flagged by variable f_r): the temperature is increased to the last "improvement temperature" t_{imp} (the temperature at which the last better solution was found) and the search starts from the best solution found so far. The temperature continues to increase according to the function $t = t/(1 - \beta t)$ until an improved solution is found. The system is then switched to the "annealing" phase and the temperature begins to decrease again. Therefore, in this algorithm, the temperature decreases gradually and frequently. However, once the system gets stuck at a local optimum, the temperature increases very quickly to escape from the local optimum.
- The weights of the low-level heuristics are updated after every learning period and normalised by their acceptance ratios $\left(c_i^{accept}/c_i^{total}\right)$ during the "annealing" phase and by ratios c_i^{new}/c_i^{total} during the "reheating" phase. At each iteration, a low-level heuristic is selected with probability $p_i = w_i/\sum_{i=1}^{n} w_i$, which is similar to the stochastic ranking method used in the evolutionary algorithm in Runarsson and Yao (2000).

3.2 Compared with other relevant meta-heuristics

The proposed simulated annealing hyper-heuristic (SAHH) is closely related to some existing metaheuristics, in particular, iterative local search (ILS) (Lourenco et al. 2003) and adaptive large neighbourhood search (ALNS) (Ropke and Pisinger 2006). We now briefly compare our proposed SAHH against ILS and ALNS.

3.2.1 SAHH versus ILS

Like SAHH, the basic iterated local search is simple and easy to implement. An ILS method contains two iteratively executed phases: a solution perturbation phase and a local search phase. The main idea is to repeatedly find the local optima at different regions of the solution space. The choices of the perturbation strategy and the local search method are the key components to the success of ILS. ILS is very similar to variable neighbourhood search (VNS) (Hansen et al. 2008) except that ILS is more general in the sense that the perturbation in ILS is more flexible. Both ILS and VNS make use of multiple neighbourhoods which are explored in a pre-defined sequence. Furthermore, at each iteration, often the entire neighbourhood is explored and the best solution is returned and accepted. However, in our proposed SAHH approach, only a part of a neighbourhood is sampled and the selection of neighbourhoods/heuristics at each iteration is dynamically determined based on a reinforcement learning mechanism. In addition, a simulated annealing acceptance criterion is used in SAHH, instead of the simple acceptance criteria used in ILS and VNS. Finally, the outcome of the SA

acceptance criterion in SAHH is fed into the heuristic selection phase to continuously improve heuristic selection in the future.

3.2.2 SAHH versus ALNS

In some ways, ALNS is very similar to SAHH. For example, both methods make use of multiple heuristics to improve the current solution. The learning mechanisms in both methods are based on the ideas of reinforcement learning. The main difference between SAHH and ALNS is twofold. Firstly, SAHH emphasizes the importance and significance of the SA acceptance criterion as opposed to other simple acceptance criteria (e.g. improvement-only acceptance and all-moves acceptance). Secondly, the heuristics (or neighbourhood functions) used in SAHH and ALNS are very different. ALNS primarily uses *ruin-and-recreate* heuristics to allow for the search oscillating between feasible and infeasible regions of the search space as these regions are often where high quality solutions can be found. While in SAHH, the feasibility of the incumbent solution is always maintained and the search is only carried out in the feasible solution space. It is difficult to compare the performance between them as it depends on the structure of the problem and how these heuristics are designed for each problem. In general, ruin-and-recreate heuristics may be more suitable for handling problems with challenging constraints (e.g. constraints that lead to most neighbourhood moves being infeasible) while heuristics used in SAHH may be more applicable for problems with large, but relatively less constrained, search spaces. Furthermore, SAHH aims to relieve users and practitioners from tedious, complex programming and advocates using simple, intuitive heuristics. Designing and implementing effective ruin-and-recreate heuristics requires profound understanding of the structures of the problem and constraints. When the problem changes, which is often the case as we mentioned earlier, much more work is required to adapt ALNS to the new environment than SAHH does.

3.3 Parameter settings

In the next section we will demonstrate the high level of generality of the proposed methodology by experimenting on two very different optimisation problems: course timetabling, and bin packing. The search spaces of the course timetabling and bin packing problem are very different. Course timetabling generally has a very large search space with difficult constraints (Chiarandini and Stutzle 2003) while bin packing tends to have an "unfriendly" plateau-like search space with relatively easy constraints (Falkenauer 1996). The key contribution to the literature is that we have developed a methodology which can operate on very different problems without the high level of human development and "tailoring" that is usually required for meta-heuristic approaches to these problems. We also demonstrate that the methodology obtains results that are competitive with special purpose algorithms that have been tailored for just one problem.

A few parameters are required to be set. To obtain generality, we want to minimise the number of parameters that are required to be tuned by a user; if we have to

include some user parameters, they should be less sensitive to the changes of problem instances or domains. Let us assume that there are n low-level heuristics. The parameter settings for the simulated annealing hyper-heuristic are defined as follows and will be the same across both applications: $r_s = 0.1, r_e = 0.005, nrep = n, w_{\min} = \min\{100n/K, 0.1\}, LP = \max\{K/500, n\}$. The initial and stopping acceptance ratio (r_s and r_s) are set so that there are about 10% non-improving moves being accepted at the initial stage and only 0.5% at the final stage. *nrep=n* means that each heuristic is called approximately once at each temperature. The values of $w_{\min}$ and LP are based on some preliminary experiments on a subset of representative course timetabling instances (small 1, medium 1, large, and competition data instances 1, 6, 11 and 16) but will be set by the same method across both applications. Therefore, we do not need to re-tune the parameters of the proposed simulated annealing hyper-heuristic to different problems and applications, which is one of the key aims of hyper-heuristic approaches. Parameter K provides an alternative way for users to adjust the computational time by the algorithm to appropriate values that are permitted in various problem solving scenarios. For example, time-critical problem-solving should adopt a relative small K while non-time-critical problems (e.g. timetabling) could use a relatively large K to obtain high quality solutions. Therefore, in this paper K will be set to different values for different problems. All the experiments were run on a PC Pentium IV 1.8GHZ with 256 MB RAM running Microsoft Windows XP Professional.

4 Computational experiments

4.1 An application to university course timetabling

4.1.1 Problem description

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. For the purpose of comparison, we shall use the same model presented in Rossi-Doria et al. (2002) and Socha et al. (2002). This model provides a representation of a typical course timetabling problem and has been widely used in recent publications (McMullan 2007). The model formulates the problem as follows:

Given a set of events e_i ($i = 0, \ldots, E$) and a number of rooms r_j ($j = 0, \ldots, R$), with each room having F types of features. Each event is attended by a given number of students and requires some of the room features. The total number of students is S. The aim of the problem is to assign every event e_i to a timeslot t_k ($k = 1, \ldots, 45$) and a room r_j so that the following hard constraints are satisfied:

- No student should be assigned to more than one event in a timeslot;
- 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 timeslot.

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

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

We adopt the same solution representation that was used in both Socha et al. (2002) and Burke et al. (2003b). 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 timeslots assigned to the given events. Again, similar to Socha et al. (2002) and Burke et al. (2003b), room assignments are dealt with separately by using a matching algorithm. A fast pattern-based objective evaluation method was also used [see Burke et al. (2003a) for more details].

4.1.2 Low-level heuristics

The three simple low-level heuristics that we use are as follows:

H1. Shift: move a random event from its current timeslot to another random timeslot.
H2. Swap event: swap the timeslots of two random events.
H3. Swap timeslot: swap all events of two randomly selected timeslots.

Note that the above low-level heuristics only operate on feasible solutions. The current solution is returned if an infeasible solution is generated. All three heuristics are relatively easy to implement and are much simpler than those neighbourhood structures that were used in other approaches, for example, Abdullah et al. (2007).

4.1.3 Computational results

The proposed simulated annealing hyper-heuristic algorithm was tested on two course timetabling benchmark data sets. The first data set was originally used in Socha et al. (2002). It consists of five small instances ($E = 100, S = 80, R = 5F = 5$), five medium instances ($E = 400, S = 200, R = 10F = 5$) and one large instance ($E = 400, S = 400, R = 10F = 10$). The second data set is drawn from the International Timetabling Competition organised by the Metaheuristic Network (2003). This data set contains twenty problem instances. The parameters of these instances are as follows: $\in E\,[350, 440]\,, S \in [200, 350]\,, R \in [10, 11]\,, F\,[5, 10] \in$. The proposed simulated annealing hyper-heuristic starts from a feasible initial solution which was constructed by a simple and quick hybrid heuristic procedure similar to the approach proposed in Asmuni et al. (2005). However, to quickly obtain a feasible initial solution, which can be used by our simulated annealing hyper-heuristic, the hybrid heuristic procedure used in this paper stops as soon as a feasible solution is found. The number of iterations for SAHH is set as follows: for the first data set, the initial experiment is set to a relatively small number of iterations $K = 500{,}000$, which corresponds to less than 1 min of computational time for small instances and 1.5 min for medium and large instances (see Table 1 for details). Furthermore, since computational time is generally considered to be non-critical for the course timetabling problem, a larger number of iterations $\left(K = 2 * 10^7\right)$ were also investigated for this data set. For the second data set (the competition data set), we set $K = 2 * 10^7$ which corresponds

Table 1 A comparison of SAHH with other approaches for the course timetabling problem on the first data set

Datasets	VNS$^+$ (Abdullah et al. 2007)		TSHH (Burke et al. 2003b)		GHH (Qu and Burke 2009)		SAHH ($K = 500{,}000$)			SAHH $\left(K = 2 * 10^7\right)$		
	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Time (s)	Best	Mean	Time (s)
Small1	**0**	**0**	1	2.2	**0**	0.2	**0**	0.6	42.2	**0**	**0**	463.7
Small2	**0**	**0**	2	3.0	**0**	0.6	**0**	2.2	49.1	**0**	**0**	560.2
Small3	**0**	**0**	**0**	1.4	**0**	**0**	1	1.2	51.3	**0**	**0**	507.9
Small4	**0**	**0**	1	1.8	**0**	0.4	1	1.8	53.4	**0**	0.2	760.0
Small5	**0**	**0**	**0**	0.2	**0**	0.1	**0**	0.6	36.6	**0**	**0**	351.4
Medium1	242	245	146	179.0	257	261	102	117.0	88.4	**38**	45.8	2,498.4
Medium2	161	162.6	173	197.6	259	273	114	122.0	86.9	**28**	36.2	2,462.9
Medium3	265	267.8	267	295.4	192	241.5	125	150.2	86.4	**48**	54.2	2,539.6
Medium4	181	183.6	169	180.0	235	242	106	110.6	86.5	**21**	27.0	2,480.2
Medium5	151	152.6	303	388.5	112	116	106	143.2	81.1	**12**	18.2	2,497.3
Large	n.a.	n.a.	1,166	1,166.0	1,132	1,135	653	670.2	89.0	**519**	569.2	2,877.9

to between 45 and 55 min of computational time on the same machine. All the other parameters remain unchanged. In a similar way to the research undertaken by Burke et al. (2003a,b), five independent runs were carried out for each problem instance using different random seeds and both best and average results are reported. Note that although the total number of iterations for SAHH is larger than the tabu search hyper-heuristic (TSHH) in Burke et al. (2003b), all three low-level heuristics used in this paper are much faster than some of the low-level heuristics used in TSHH. For example, one of the low-level heuristics used in TSHH is defined as "1st improving soft constraints without worsening hard constraints". A single run of this heuristic usually involves evaluating several neighbouring solutions, or even the whole neighbourhood before finding an improved solution. Therefore, it can be much more computationally expensive than the low-level heuristics used in SAHH.

Table 1 presents a comparison between SAHH and other meta-heuristic and hyper-heuristic approaches reported in the literature for the first data set instances, including a tabu search hyper-heuristic (TSHH) (Burke et al. 2003b), a variant of variable neighbourhood search $\left(\text{VNS}^+\right)$ (Abdullah et al. 2007) and a graph-based hyper-heuristic (GHH) (Qu and Burke 2009). The computational times by VNS+ (Abdullah et al. 2007) are 50 s for small instances, and around 8 h for medium and large instances on a PC with Pentium 1.2 GHz with 256 MB RAM. The experimental environment for GHH (Qu and Burke 2009) was a PC with Pentium IV 3.0 GHz and 1 GB RAM. The computational time for small instances (respectively the medium instances and the large instance) is around 50 s (4.6, and 5.6 h respectively). Our SAHH was run on a PC Pentium IV 1.8 GHz CPU and 256 MB RAM with the computational time (when $K = 500{,}000$) ranging from 0.7 to 1.5 min. Therefore, we generally uses less computational time (particularly for the medium instances and the large instance) even taking into account the performance scales of these different machines. Both the best and average results among the 5 runs are reported in Table 1. The best results among

Table 2 The statistic values (p values) of the Friedman tests for the first course timetabling data set

	All algorithms	SAHH versus VNS+	SAHH versus TSHH	SAHH versus GHH
Small1	0.009	**0.371**	0.025	**0.655**
Small2	0.011	**0.074**	**0.371**	**0.180**
Small3	0.006	0.025	**0.180**	0.025
Small4	0.003	0.025	**0.180**	0.025
Small5	0.046	**0.371**	**0.655**	**0.655**
Medium1	0.002	0.025	0.025	0.025
Medium2	0.002	0.025	0.025	0.025
Medium3	0.002	0.025	0.025	0.025
Medium4	0.002	0.025	0.025	0.025
Medium5	0.007	**0.655**	0.025	**0.180**
Large	0.007	n.a	0.025	0.025

all the four algorithms are presented in bold. The results of VNS+ for the large instance are missing here due to unavailability.

In order to find out whether the performance of the four algorithms (SAHH with $K = 500{,}000$, VNS+, TSHH, and GHH) are different in terms of solution quality, and whether our SAHH performs better, several Friedman tests (Conover 1999) were carried out. The mean values by VNS+, TSHH, and GHH were used in the tests. Our first test included all four algorithms. The null hypothesis, therefore, is that there is no significant difference between VNS+, TSHH, GHH, and SAHH. The second column of Table 2 gives the corresponding statistic values (p values). It can be seen that this null hypothesis is rejected for every problem instance at confidence levels ranging between 0.002 and 0.046. This means that at least 2 algorithms performed significantly differently for each of these problem instances.

In the next step, we then carried out paired Friedman tests between our SAHH ($K = 500{,}000$) and each of the other three algorithms. The null hypotheses respectively are that there is no significant performance difference between SAHH and VNS+ (respectively TSHH, GHH). The corresponding p values are shown in Table 2 (columns 3–5) and large p values (i.e. >0.05) are highlighted in bold. Based on the data from both Tables 1 and 2, it can be concluded that, at the confidence level $p = 0.05$, VNS+ performed significantly better than SAHH for 2 small instances (small 3, and 4), but SAHH was significantly better for 4 medium instances (medium 1, 2, 3, and 4). For the other 4 instances, there is no significant performance difference between them. The comparison between SAHH and TSHH is more straightforward. SAHH outperformed TSHH significantly for 7 instances. For the remaining 4 small instances, there is no significant difference between them. Comparing between SAHH and GHH, we can see that the performances by SAHH and GHH are not significantly different for 4 instances (3 small instances and 1 medium instance). SAHH outperformed GHH for 5 instances (medium 1, 2, 3, 4 and large) and GHH did better than SAHH for 2 instances (small 3 and 4). In summary, it seems that the proposed simulated annealing hyper-heuristic is particularly suited to instances of larger sizes. This is probably due

Table 3 A comparison of SAHH with bespoke approaches for the course timetabling problem on the competition data set

Instances	Winner	2nd	3rd	4th	5th	6th	7th	SAHH	
								Best	Mean
1	**45**	61	85	63	132	148	178	86	96.0
2	**25**	39	42	46	92	101	103	59	68.6
3	**65**	77	84	96	170	162	156	116	125.6
4	**115**	160	119	166	265	350	399	135	162.0
5	102	161	77	203	257	412	336	196	213.8
6	13	42	**6**	92	133	246	246	11	14.4
7	44	52	**12**	118	177	228	225	**12**	18.2
8	**29**	54	32	66	134	125	210	36	43.0
9	**17**	50	184	51	139	126	154	46	49.4
10	**61**	72	90	81	148	147	153	85	95.2
11	44	53	73	65	**35**	144	169	76	93.2
12	107	110	**79**	119	290	182	219	134	140.4
13	**78**	109	91	160	251	192	248	120	134.6
14	52	93	**36**	197	230	316	267	40	56.4
15	**24**	62	27	114	140	209	235	25	41.6
16	**22**	34	300	38	114	121	132	33	42.2
17	86	114	**79**	212	186	327	313	249	280.0
18	**31**	38	39	40	87	98	107	57	78.6
19	**44**	128	86	185	256	325	309	104	119.8
20	7	26	**0**	17	94	185	185	1	7.2
Average	50.6	76.8	77.1	106.5	166.5	207.2	217.2	81.1	94.0

to the hyper-heuristic learning mechanism which learns how to bias solution sampling strategies towards more promising regions of the search space. When a longer computational time is allowed (e.g. $K = 2 * 10^7$), SAHH is able to further improve these results considerably.

Table 3 gives a comparison for the second data set between our simulated annealing hyper-heuristic and the top seven bespoke approaches in the international timetabling competition (Metaheuristic Network 2003). The best results are highlighted in bold. It can be seen that, on this data set, our simulated annealing hyper-heuristic would be ranked in fourth place both in terms of average objective value and the official ranking approach across twenty instances. Three approaches perform better in terms of solution quality. In fact, another hybrid algorithm by Chiarandini et al. (2006) would have won the competition but could not enter because they were the organisers (see their results in Table 4). This is not surprising since these bespoke approaches were specifically tailored for these problem instances and heavily utilised some of the features of these instances while our method was designed with generality in mind. For example, according to Kostuch (2004), one of the features is that all the instances are

Table 4 A comparison of SAHH with the hybrid algorithm by Chiarandini et al. (2006)

Instances	1	2	3	4	5	6	7	8	9	10
Chiarandini et al. (2006)	57	31	61	112	86	3	5	4	16	54
SAHH	86	59	116	135	196	11	12	36	46	85
Instances	11	12	13	14	15	16	17	18	19	20
Chiarandini et al. (2006)	38	100	71	25	14	11	69	24	40	0
SAHH	76	134	120	40	25	33	249	57	104	1

created in such a way that at least a "perfect" solution (i.e. solution with a zero penalty value) exists. This feature was heavily exploited by the winning algorithm in its initial solution procedure in order to obtain an initial solution being very close to an optimal solution (note that this does not necessarily mean this initial solution has a very small objective value). Therefore, these procedures may not be as effective when applied to other problem instances which do not contain these features. In fact, we have run one of the top three algorithms on the first data set and some results are far worse than our results on these instances. This indicates that although these bespoke algorithms perform very well on the competition data set, their performance can decrease significantly even when applied to different instances from the same domain.

4.2 An application to bin packing

In the previous section, we have shown that our proposed simulated annealing hyper-heuristic is able to improve upon the performance of a recently proposed tabu search hyper-heuristic. In fact, it also produced better results than some problem specific bespoke meta-heuristic approaches. In this section, we apply the proposed algorithm to the well-known bin-packing problem to further test its generality across different problem solving environments. We will use exactly the same parameter settings as in the previous experiment. As before, we only need to change the initial solution, the evaluation function and the set of low-level heuristics.

4.2.1 The bin packing problem

The one dimensional bin packing problem is defined as follows. Given a set of items I ($i = 1, \ldots, n$) each having an associated size or weight w_i, and a set of bins with identical capacities c, the problem is to pack all the items into as few bins as possible, without exceeding the capacity of the bins. The bin packing problem is a well-known NP-hard combinatorial optimisation problem (Martello and Toth 1990a). However, it is not difficult to get a lower bound of the problem. A straightforward lower bound can be obtained by $L_1 = \sum_{i=1}^{n} w_i/c$. Some stronger lower bounds were studied in Martello and Toth (1990b), Scholl et al. (1997). Considerable research has been carried out on exact methods. One of the most successful algorithms for bin packing is known as the Martello-Toth Procedure. This is a branch and bound based exact method originally proposed in Martello and Toth (1990b). Some other versions of exact methods

are proposed in Scholl et al. (1997), Belov and Scheithauer (2006). Heuristic based approaches have also been studied. Apart from some well-known constructive heuristics (for example, First-Fit-Descent and Best-Fit-Descent), meta-heuristic approaches have been adapted for the bin packing problem, including a grouping genetic algorithm (Falkenauer 1996) and variable neighbourhood search (Fleszar and Hindi 2002). In both applications, transformed objective functions were used because if the number of used bins is chosen as the objective function, the bin packing problem solution space is fairly flat and a large number of solutions correspond to the same objective value. General local search approaches cannot be well guided by this objective function (Falkenauer 1996, 1998). Recently, Alvim et al. (2004) proposed a hybrid algorithm which combines several strategies, including a dual min-max method for generating initial solutions, new dominance criteria, unbalancing heuristic, etc. Superior results were obtained by the algorithm compared with other existing heuristic approaches. However, the algorithm is highly problem-specific and is difficult to adapt to other problems with different solution structures. For instance, one may want to use different objective functions (instead of the number of bins), or have some additional constraints, which would probably lead to these strategies being invalid or ineffective. Ross et al. (2003) proposed a genetic algorithm hyper-heuristic for the problem. The algorithm was firstly trained on a subset of benchmark problems and after the training, the fittest chromosome was then applied to every benchmark problem. The computational results demonstrate that the evolved algorithm performs better than any of the individual low-level heuristics.

4.2.2 Problem specific input

Initial solution and evaluation function

Our initial solution is created by a time bounded relaxed Minimum Bin Slack (MBS) heuristic that is similar to the MBS heuristic used in Fleszar and Hindi (2002) except that the stopping criteria is relaxed by allowing a small slack value (equal to the average slack value allowed in the lower bound L_1) and 0.02 s computational time limit. This modification could let the heuristic run much faster without compromising its performance. We use the original objective function (i.e. the number of used bins) rather than using some of the transformed evaluation functions as in Falkenauer (1996), Scholl et al. (1997), Fleszar and Hindi (2002).

Low-level heuristics

A total of five heuristics are used, which are described as follows:

H1. Shift: this heuristic selects each item from the bin with the largest residual capacity and tries to move the items to the rest of the bins using the best fit descent heuristic.
H2. Split: this heuristic simply moves half the items (randomly selected from) the current bin to a new bin if the number of items in the current bin exceeds the average item numbers per bin.

H3. Exchange largestBin_largestItem: this heuristic selects the largest item from the bin with the largest residual capacity and exchanges this item with another smaller item (or several items whose capacity sum is smaller) from another randomly selected non-fully-filled bin. The idea behind this heuristic is to transfer smaller residual capacity from a random bin to a bin with the largest residual capacity so that this bin can be emptied by other heuristic(s).

H4. Exchange randomBin_reshuffle: the idea behind this heuristic is similar to H3, which attempts to transfer residual capacity to the bins with larger residual capacity. This heuristic randomly selects two non-fully-filled bins. The probability of the selection is proportional to the amount of their residual capacities. All items from these two bins are then considered and the best items' combination is identified so that it can maximally fill one bin. The remaining items are filled into the other bin.

H5. BestPacking: this heuristic firstly selects the biggest item from a probabilistically selected bin. The time bounded relaxed MBS heuristic is then used to search for a good packing that contains this item and considers all the other items (the sequence of these items is sorted by the residual capacity of the corresponding bins with ties broken arbitrarily). All the items that appeared in the packing found by time bounded relaxed MBS are then transferred into a new bin. Again, the time limit is set to a small value for quick implementation of the heuristic (0.02 s in this case). The probability of selecting a bin is calculated by $\psi = \frac{resCap_i}{\sum resCap_i}$, where $resCap_i$ is the residual capacity of the bin i. Hence, the selection is in favour of the bins with the larger residual capacity.

Note that all of the heuristics outlined above will guarantee to output feasible solutions (the current solution is returned if the new candidate solution is infeasible). All of the heuristics are simple, straightforward and easy to implement.

4.2.3 Bin packing benchmark problems

Three sources (groups) of benchmark problems are available from the literature for the one dimensional bin packing problem. One of them is from the OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html). This group of data sets consists of two classes of problems: uniform and triplet. In the uniform class, the number of items is 120, 250, 500 and 1,000 respectively and their sizes are uniformly distributed in the range of [20,100]. The bin capacity is 150. We shall denote these by Fal_U120, Fal_U250, Fal_U500 and Fal_U1000 respectively. There are 20 instances for each problem size and hence 80 problem instances in total. In the triplet class, the bin capacity is 1,000 and the item sizes are deliberately generated such that, in the optimal solution, every bin contains exactly three items (one "big" and two "small") without any residual capacity. The number of the items is 60, 120, 249 and 501 (denoted by Fal_T60, Fal_T120, Fal_T249 and Fal_T501 respectively) and each of them contains 20 instances. This class of data sets is claimed to be more difficult because of the fact that no residual capacity is allowed in any bin in the optimal solution. In Fleszar and Hindi (2002), 1,000 runs of perturbation MBS were implemented before applying their variable neighbourhood search algorithm in order to solve this class of

data sets. However, in this paper, we allow the simulated annealing hyper-heuristic to automatically adapt to different classes of problem instances by choosing different heuristics.

The second group of data sets was generated and studied by Scholl et al. (1997) and is available at http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm. It contains three sets (denoted by Sch_Set1, Sch_Set2 and Sch_Set3 respectively) and comprises a total of 1,210 instances. The lower bounds of these instances are available on the same website. The parameters to create Sch_Set1 and Sch_Set2 include the number of the items (ranging from 50 to 500), bin capacity and the ranges that the items' sizes are drawn from Sch_Set1 consists of 720 problem instances and the expected average number of items per bin is no larger than three. However, in Sch_Set2, the average number of items per bin varies between three, five, seven and nine. The data sets Sch_Set3 are considered to be harder problem instances because the item size is drawn from a very large range such that no two items have the same size. Only 10 instances are included in this set.

The third group of benchmark data sets are maintained by the EURO Special Interest Group on Cutting and Packing and are downloadable from (http://www.apdio.pt/sicup/index.php). This group of data sets represents a collection of difficult instances from a large number of test instances in three publications (Waescher and Gau 1996; Schwerin and Wascher 1997; Belov and Scheithauer 2006). The data sets Sch_Wae1 and Sch_Wae2 consist of 200 instances selected from (Schwerin and Wascher 1997) which are particularly hard for First-Fit-Descent and partially hard for the Martello-Toth Procedure (in fact, none of these instances can be solved optimally by First-Fit-Descent). The data sets Wae_Gau1 are collections of 17 instances from (Waescher and Gau 1996) which have not yet been solved by either First-Fit-Descent or Martello-Toth Procedure. Note that all the instances in data sets Wae_Gau2 also appear in Wae_Gau1 so they are considered in this paper. The lower bounds of these instances are available from http://www.inf.puc-rio.br/~alvim/adri\discretionary-ana/_les/detbp.pdf.

4.2.4 Computational results

The low-level heuristics and initial solutions were input to our simulated annealing hyper-heuristic framework which was run on a PC with Intel Core 2 CPU 1.8GHz and 2GB RAM running Windows XP, but the computation was limited to one processor only. The algorithms were run 20 times for each instance, using a different random seed each time. For each problem instance, the simulated annealing hyper-heuristic stops either after the number of iterations reaches K or when the computational time exceeds a given limited t. Therefore, we can use both K and t to control the amount of computational time permitted for the SAHH. To compare the performance of the different algorithms, the following symbols are used:

- #num: the number of instances in the given data sets.
- #opt: the number of instances for which the given algorithm finds a solution with the lower bound objective value (i.e. the algorithm has solved those instances optimally). For the algorithm SAHH, the average values over 20 runs were reported. For

Table 5 A comparison with perturbation MBS' + VNS (Fleszar and Hindi 2002)

Data sets	#num	Perturbation MBS' + VNS				SAHH			
		#opt	Max abs.	Av. cpu	Max cpu	#opt	Max abs.	Av. cpu	Max cpu
Fal_U120	20	**20**	0	0.02	0.04	**20.0**	0	0.00	0.19
Fal_U250	20	**19**	1	0.03	0.16	17.5	1	0.15	1.02
Fal_U500	20	**20**	0	0.04	0.14	19.0	1	0.07	1.00
Fal_U1000	20	**20**	0	0.07	0.27	**20.0**	0	0.01	0.22
Fal_T60	20	**20**	0	0.01	0.01	19.9	1	0.03	1.00
Fal_T120	20	**20**	0	0.02	0.04	**20.0**	0	0.04	1.00
Fal_T249	20	**20**	0	0.02	0.04	**20.0**	0	0.04	1.00
Fal_T501	20	**20**	0	0.06	0.10	19.9	1	0.05	0.86
Sch_Set1	720	694	2	0.15	1.78	**697.5**	1	0.04	1.02
Sch_Set2	480	474	1	0.10	4.57	**473.4**	1	0.03	1.02
Sch_Set3	10	2	1	3.74	5.05	**7.3**	3	0.35	1.02
All	1,370	1,329	2	0.14	5.05	**1,334.3**	3	0.07	1.02

the meta-heuristic approach, perturbation MBS' + VNS (Fleszar and Hindi 2002), only single run results are reported due to no average results being available.

- max abs.: the maximal absolute deviation from the optimal solution or the best known lower bound if the optimal solution is not known.
- av. cpu: the average CPU time spent for the given data sets (in seconds).
- max cpu: the maximal CPU time spent for an instance in the given data sets (in seconds).

We choose to compare our algorithm against two state-of-the-art bin packing heuristics, a variable neighbourhood search based approach (MBS'+VNS) proposed in Fleszar and Hindi (2002) and a hybrid procedure (HP_BP) in Alvim et al. (2004). Perturbation MBS'+VNS was run on a PC with 400 MHz Pentium II CPU running Windows NT4.0, while HP_BP was tested on a 1.7 GHz Pentium IV PC with 256 MB RAM. In order to let our proposed algorithm having comparable computational power when they are compared against these two approaches from the literature, we scaled our computational time and run SAHH with settings: ($K = 150{,}000$, $t = 1$ s) and ($K = 1{,}600{,}000$, $t = 40$ s). These two settings are based on the performance scaling information of different computers available from http://www.roylongbottom.org.uk/linpack%20results.htm. That is, the CPU time limit for SAHH was 1 s when compared against the hybrid VNS method in Fleszar and Hindi (2002) and 40 s when compared with the HP_BP approach in Alvim et al. (2004). Note that the computational time limits here are applied to SAHH and do not include the time for the generation of an initial solution where SAHH starts the search. Tables 5 and 6 respectively present a comparison of our simulated annealing hyper-heuristic with Perturbation MBS' + VNS and HP_BP (the computational results of Perturbation MBS' + VNS on the third group of data sets, Sch_Wae1, Sch_Wae2, Wae_Gau1, are not available in the literature). Numbers in bold represent the best results. It can be seen that our simulated

Table 6 A comparison with the hybrid procedure in Alvim et al. (2004)

Data sets	#num	HP_BP				SAHH			
		#opt	Max abs.	Av. cpu	Max cpu	#opt	Max abs.	Av. cpu	Max cpu
Fal_U120	20	**20.0**	0	0.00	0.01	**20.0**	0	0.00	0.13
Fal_U250	20	**20.0**	0	0.15	3.19	19.0	1	2.09	39.74
Fal_U500	20	**20.0**	0	0.00	0.01	19.1	1	1.92	40.02
Fal_U1000	20	**20.0**	0	0.01	0.03	**20.0**	0	0.01	0.11
Fal_T60	20	**20.0**	0	0.33	2.53	20.0	0	0.03	0.56
Fal_T120	20	**20.0**	0	1.14	6.88	**20.0**	0	0.03	0.42
Fal_T249	20	**20.0**	0	0.29	2.91	**20.0**	0	0.03	0.39
Fal_T501	20	**20.0**	0	1.24	19.26	20.0	0	0.05	0.52
Sch_Set1	720	**719.2**	1	0.20	23.89	708.8	1	0.65	40.03
Sch_Set2	480	**480.0**	0	0.01	1.89	475.7	1	0.38	40.03
Sch_Set3	10	**10.0**	0	4.71	50.61	**8.0**	1	8.10	40.03
Sch_Wae1	100	**100.0**	0	0.02	0.14	**100.0**	0	0.19	32.41
Sch_Wae2	100	**100.0**	0	0.02	3.58	99.5	1	0.36	40.05
Wae_Gau1	17	**12.0**	1	0.60	2.40	**12.0**	1	11.77	40.03
All	1,587	**1,581.2**	1	0.38	50.61	1,561.9	1	1.83	40.05

annealing hyper-heuristic was slightly inferior to the hybrid VNS approach for the first group of data sets but did better for the second group of data sets in terms of solution quality. Overall, the simulated annealing hyper-heuristic (which, recall, is not specifically designed for the bin packing problem) solved around 5 more instances to optimality than the hybrid VNS. Note that our simulated annealing hyper-heuristic is generally slower than tailor-made meta-heuristics since the hyper-heuristic needs to spend time to learn about the nature of the problem and then adapt to that problem. In the next section, it can be seen that better results can be obtained when more computational time is permitted. However, according to Fleszar and Hindi (2002), results by Perturbation MBS' + VNS failed to improve noticeably even when the computational time was tripled.

Compared with the current best bin packing heuristic approach HP_BP (Alvim et al. 2004), the simulated annealing hyper-heuristic is slightly worse in terms of solution quality (see Table 6). In fact, HP_BP performed excellently and achieved around 1581 optimal solutions out of 1,587 problem instances. Our simulated annealing hyper-heuristic solved 1,561.9 out of 1,587 (98.4%) instances to optimality on average. For the remaining 1.6% of the instances, the solution obtained by the simulated annealing hyper-heuristic is only one bin away from the lower bounds. The slightly better performance by HP_BP does not come as a surprise as it draws heavily upon several problem-specific features, such as the reduced procedure MTRP, the dual min-max problem (BPP-2), etc. It would be very difficult to apply HP_BP to other problems (such as course timetabling for example). However, with the slight decrease in solution quality, the simulated annealing hyper-heuristic is able to gain a much higher level of

generality and reusability. As shown throughout this paper, one can conveniently adapt the simulated annealing hyper-heuristic to a very different problem without significant effort and by drawing only upon a small set of simple low-level heuristics.

Our simulated annealing hyper-heuristic has also shown significant improvement over a previously proposed genetic algorithm hyper-heuristic (Ross et al. 2003) for the bin packing problem. In Ross et al. (2003), the proposed hyper-heuristic managed to solve 80% of instances to optimality. However, as shown in Tables 5 and 6, the proposed simulated annealing hyper-heuristic is able to solve around 98% of instances to optimality, on average. The main advantage of the genetic algorithm hyper-heuristic for the bin packing is that the algorithm is constructive. That is, once a heuristic is evolved, it can be used to solve many instances with less computational time than the simulated annealing hyper-heuristic in this paper.

Across several data sets, our simulated annealing hyper-heuristic exhibits very good performance considering that it is not specifically designed for this problem and, indeed, the main purpose of designing it was to develop a methodology which could easily be applied to a number of different problems. As stated in Sect. 3.3, all the parameters (except the total number of iterations) used in the simulated annealing hyper-heuristic are the same for both problems and problem instances.

5 Conclusions

This paper has proposed a new simulated annealing hyper-heuristic methodology as a generic framework for search problems. The proposed hyper-heuristic adopted the two layer framework of previous hyper-heuristics which helps to improve the generality of the algorithm but we have incorporated three new features: a simulated annealing acceptance criterion, a stochastic heuristic selection strategy and a short-term memory. The simulated annealing acceptance criterion helps to improve the acceptance decisions of heuristic moves and it also provides useful feedback for better heuristic selections in the future. A stochastic heuristic selection strategy was employed in this hyper-heuristic, in preference to the deterministic heuristic selection methods used in most current hyper-heuristic approaches that have been reported in the literature. A short-term memory underpins the heuristic selection mechanism. The underlining rationale is that the search proceeds within a dynamic environment which may change during different stages of the search and the information gathered during the initial stages may be not useful later in the search.

To demonstrate the increased level of generality and competitive performance of the proposed simulated annealing hyper-heuristic, we have applied the algorithm to two very different and challenging search problems with the same parameter settings across two problem domains: university course timetabling and bin packing. Experimental results on a large number of benchmark data sets show that the proposed simulated annealing hyper-heuristics provide considerable improvement on a previously proposed tabu search hyper-heuristic. In fact, the simulated annealing hyper-heuristic also produced better results than several recently proposed problem-specific meta-heuristic approaches which do not have the significant advantage of having a higher level of generality. This paper contributes to the literature with a more efficient hyper-heu-

ristic search technique which can be readily applied to different applications and which can still produce high quality solutions that are either competitive with or (sometimes) better than bespoke problem specific methods. Almost all of the heuristics which have appeared in the scientific literature have been designed and chosen by humans. The work described in this paper provides a step toward the goal of understanding how we can automate the heuristic design process and begin to design computer systems which can select and design heuristics.

References

Abdullah S, Burke EK, McCollum B (2007) Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In: Doerner KF, Gendreau M, Greistorfer P, Gutjahr G, Hartl RF, Reimann M (eds) Metaheuristics—progress in complex systems optimization. Springer, New York, pp 153–169

Alvim ACF, Ribeiro CC, Glover F, Aloise DJ (2004) A hybrid improvement heuristic for the one-dimensional bin packing problem. J Heuristics 10:205–229

Asmuni H, Burke EK, Garibaldi JM (2005) Fuzzy multiple heuristic ordering for course timetabling. In: Proceedings of the 5th United Kingdom workshop on computational intelligence (UKCI05), London, UK, pp 302–309

Bai R, Kendall G (2005) An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solver—(Operations research/computer science interface series, vol 32). Springer, New York, pp 87–108

Belov G, Scheithauer G (2006) A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. Eur J Oper Res 171(1):85–106

Burke EK, Kendall G (2005) Search methodologies: introductory tutorials in optimization and decision support techniques. Kluwer, Dordrecht

Burke EK, Bykov Y, Newall J P, Petrovic S (2003a) A time-predefined approach to course timetabling. Yugosl J Oper Res 13(2):139–151

Burke EK, Kendall G, Soubeiga E (2003b) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9:451–470

Burke EK, Petrovic S, Qu R (2006) Case based heuristic selection for timetabling problems. J Sched 9(2):115–132

Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for timetabling problems. Eur J Oper Res 176(1):177–192

Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR (2009) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheursistics. Springer, New York, pp 449–468

Burke EK, Hyde M, Kendall G, Woodward JR (2010) A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. IEEE Trans Evol Comput 14(6):942–958

Chiarandini M, Birattari M, Socha K, Rossi-Doria O (2006) An effective hybrid algorithm for university course timetabling. J Sched 9(5):403–432

Chiarandini M, Stutzle T (2003) A landscape analysis for a hybrid approximate algorithm on a timetabling problem. TU Darmstadt Technical Report, AIDA-03-05

Conover WJ (1999) Practical nonparametric statistics, 3rd edn. Wiley, New York

Cowling P, Kendall G, Soubeiga E (2001) A hyperheuristic approach to scheduling a sales summit. In: Burke EK, Erben W (eds) Selected papers of the 3rd international conference on the practice and theory of automated timetabling, Lecture Notes in computer science series, vol 2079. Springer, pp 176–190

Crowston WB, Glover F, Thompson GL, Trawick JD (1963) Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research Memorandum, GSIA, Carnegie Mellon University, Pittsburgh, p 117

Dowsland KA, Soubeiga E, Burke EK (2006) A simulated annealing hyper-heuristic for determining shipper sizes. Eur J Oper Res 179(3):759–774

Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. J Heuristics 2:5–30
Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, New York
Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall, Englewood Cliffs, NJ, pp 225–251
Fleszar K, Hindi KS (2002) New heuristics for one-dimensional bin-packing. Comput Oper Res 29(7):821–839
Glover F, Kochenberger G (2003) Handbook of metaheuristics. Kluwer, Dordrecht
Han L, Kendall G (2003) Guided operators for a hyper-heuristic genetic algorithm. In: AI 2003: advances in artificial intelligence: the proceedings of 16th Australian conference on AI. Lecture notes in computer science, vol 2903. Springer, pp 807–820
Hansen P, Mladenovic N, Moreno Perez JA (2008) Variable neighbourhood search: methods and applications. 4OR-A Q J Oper Res 6:319–360
Hart E, Ross P, Nelson JA (1998) Solving a real-world problem using an evolving heuristically driven schedule builder. Evol Comput 6(1):61–80
Kitano H (1990) Designing neural networks using genetic algorithms with graph generation system. Complex Syst 4:461–476
Kostuch P (2004) The university course timetabling problem with a 3-phase approach. In: Burke EK, Trick M (eds) The practice and theory of automated timetabling V, Lecture notes in computer science, vol 3616. Springer, Berlin, pp 109–125
Lourenco HR, Martin OC, Stutzle T (2003) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Dordrecht, pp 321–354
Lundy M, Mees A (1986) Convergence of an annealing algorithm. Math Program 34:111–124
Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, New York
Martello S, Toth P (1990) Lower pounds and reduction procedures for the bin packing problem. Discret Appl Math 28:59–70
McMullan P (2007) An extended implementation of the Great Deluge Algorithm for course timetabling. Lecture notes in computer science, vol 4487. Springer, Berlin, pp 538–545
Metaheuristic Network (2003) International timetabling competition: Competition results http://www.idsia.ch/Files/ttcomp2002/results.htm. Accessed 11 October 2010
Mockus J (1989) Bayesian approach to global optimization. Kluwer, Dordrecht
Mockus J (1994) Application of bayesian approach to numerical methods of global and stochastic optimization. J Glob Optim 4(4):347–366
Mockus J (1997) Bayesian heuristic approach to discrete and global optimization. Kluwer, Dordrecht
Mockus J (2000) A set of examples of global and discrete optimization: application of bayesian heuristic approach. Kluwer, Dordrecht
Nareyek A (2003) Choosing search heuristics by non-stationary reinforcement learning. In: Resende MGC, de Sousa JP (eds) Metaheuristics: computer decision-making. Kluwer, Dordrecht, pp 523–544
Qu R, Burke EK (2009) Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. J Oper Res Soc 60:1273–1285
Rattadilok P, Gaw A, Kwan RSK (2005) Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke EK, Trick M (eds) Selected papers from the 5th international conference on the practice and theory of automated timetabling. Lecture notes in computer science series, vol 3616. Springer, Berlin, pp 51–70
Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472
Ross P (2005) Hyper-heuristics. In: Burke EK, Kendall G (eds) Search methodologies: introductory tutorials in optimization and decision support techniques. Springer, Berlin, pp 529–556
Ross P, Marin-Blazquez JG, Schulenburg S, Hart E (2003) Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyper-heuristics. In: Proceeding of the genetic and evolutionary computation conference, GECCO 2003. Springer, Berlin, pp 1295–1306
Rossi-Doria O, Blum C, Knowles J, Samples M, Socha K, Paechter B (2002) A local search for automated timetabling. In: Proceedings of the 4th international conference on the practice and theory of automated timetabling [PATAT 2002] (pp 124–127)
Runarsson TP, Yao X (2000) Stochastic ranking for constrained evolutionary optimization. IEEE Trans Evol Comput 4(3):344–354

Scholl A, Klein R, Jurgens C (1997) BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Comput Oper Res 24(7):627–645

Schwerin P, Wascher G (1997) The bin-packing problem: a problem generator and some numerical experiments with FFD packing and MTP. Int Trans Oper Res 4(5/6):377–389

Socha K, Knowles J, Samples M (2002) 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, pp 1–13

Soubeiga E (2003) Development and application of hyperheuristics to personnel scheduling. PhD Thesis, The University of Nottingham, UK

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge, MA

Terashima-Marin H, Ross P, Valenzuela-Rendon M (1999) Evolution of constraint satisfaction strategies in examination timetabling. In: Proceedings of the genetic and evolutionary computation conference, GECCO 1999. Morgan Kaufmann, Los Altos, CA, pp 635–642

Venkatraman S, Yen GG (2005) A generic framework for constrained optimization using genetic algorithms. IEEE Trans Evol Comput 9(4):424–435

Waescher G, Gau T (1996) Heuristics for the integer one-dimensional cutting stock problem: a computational study. OR Spektrum 18:131–144