School of Computer Science and Information Technology
University of Nottingham
Jubilee Campus
NOTTINGHAM NG8 1BB, UK

Computer Science Technical Report No. NOTTCS-TR-2007-8

A Simulated Annealing Hyper-heuristic Methodology for Flexible Decision Support

Ruibin Bai, Jacek Blazewicz, Edmund K Burke
Graham Kendall and Barry McCollum

First released: 2007

A Simulated Annealing Hyper-heuristic Methodology for Flexible Decision Support

Ruibin Bai[1], Jacek Blazewicz[2], Edmund K Burke[1], Graham Kendall[1], Barry McCollum[3]

[1] School of Computer Science & IT, University of Nottingham
Nottingham, NG8 1BB, UK; e-mail: {rzb|ekb|gxk}@cs.nott.ac.uk

[2] Poznan University of Technology, Institute of Computing Science
60-965 Poznan, Poland; email: jblazewicz@cs.put.poznan.pl

[3] Department of Computer Science, Queen's University Belfast
Belfast, BT7 1NN, UK; e-mail: b.mccollum@qub.ac.uk

One of the main motivations for investigating hyper-heuristic methodologies is to provide a more general search framework than is currently available. Most of the current search techniques represent approaches that are largely adapted for specific search problems (and, in some cases, even specific problem instances). 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 more generally applicable to more problems. One of our motivating goals is to underpin the development of more flexible search methodologies that can be easily and automatically employed on a broader range of problems than is currently possible. Almost all the heuristics that have appeared in the literature have been designed and selected by humans. 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 data sets drawn from three very different and difficult (NP-hard) problems: nurse rostering, university course timetabling and one-dimensional bin packing. Experimental results show that the proposed hyper-heuristic is able to achieve significant performance improvements over a recently proposed tabu search hyper-heuristic without lowering the level of generality. We also show that our hyper-heuristic is capable of producing competitive results against bespoke meta-heuristics methods for these problems. In some cases, the simulated annealing hyper-heuristic has even obtained considerable improvements over some of the current best problem-specific meta-heuristic approaches. The contribution of this paper is to present a method which can be readily (and automatically) applied to very 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.

Key words: hyper-heuristics; simulated annealing; bin packing; course timetabling; nurse rostering
History:

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 some of the scientific achievements that will be 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 significant research attention (Burke et al. 2003, Burke, Kendall and Soubeiga 2003, Ross 2005) 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 three very different and challenging optimisation problems: nurse rostering, 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 “heuristics to choose heuristics” (Burke et al. 2003, Ross 2005). 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 (Burke et al. 2003, Ross 2005). 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 1960’s (Fisher and Thompson 1963, Crowston et al. 1963) and throughout the

1980's and 1990's (Mockus 1989, Kitano 1990, Hart, Ross and Nelson 1998). This section gives a short overview of relevant hyper-heuristic methods. The interested reader can refer to (Burke et al. 2003, Soubeiga 2003, Ross 2005) for more detailed overviews.

It is possible to broadly categorise hyper-heuristics into those which are *constructive* and those which represent *local search* methods. 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, Ross and Nelson 1998, Ross et al. 2003, Burke, Petrovic and Qu 2006, Burke et al. 2007). 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, Ross and Nelson (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 2002, Ross et al. 2003, Terashima-Marin, Ross and Valenzuela-Rendon 1999).

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, Kendall and Soubeiga 2003). 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, Soubeiga and Burke 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 ant systems (Burke et al. 2005), choice functions (Cowling, Kendall and Soubeiga 2001, Rattadilok et al., 2005) and case-based reasoning (Burke, Petrovic and Qu 2006).

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

3 A flexible simulated annealing 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 data. 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.

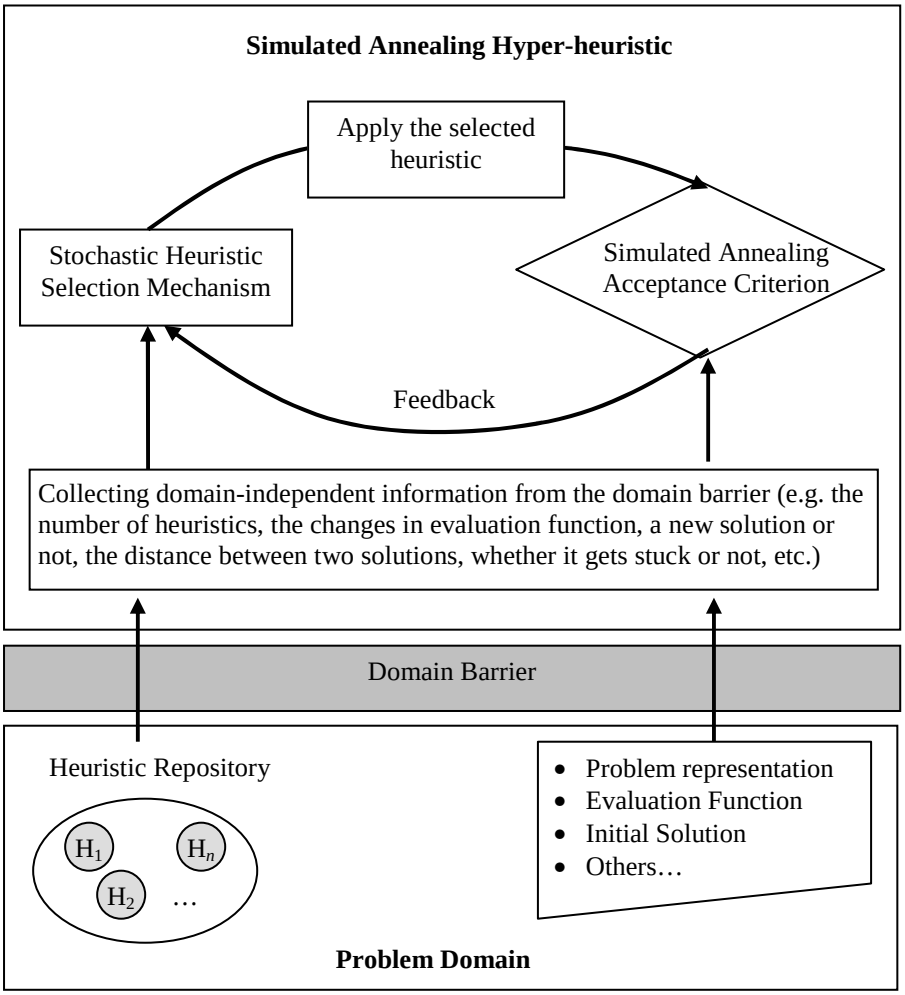

Fig. 1. The framework of our simulated annealing hyper-heuristic

Our proposed hyper-heuristic is shown in Fig. 1. It has the following features. Firstly, it adopts a simulated annealing acceptance criterion 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, Kendall and Soubeiga 2001, Nareyek 2003, Burke, Kendall and Soubeiga 2003). 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 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. In addition, the following features are also incorporated into the proposed simulated annealing hyper-heuristic:

1. 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 degrees 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 at the initial stage 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.
2. The algorithm should distinguish between the 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. Hyper-heuristics should encourage calls of the latter type of heuristic whilst reducing the first type, especially when the algorithm gets stuck at a local optimum.

In Fig. 2 we present the pseudo-code of our proposed algorithm. It assumes a minimisation problem but, of course, it is trivial to convert it to a maximisation problem. The algorithm has the following features.

- For each low-level heuristic, we associate a weight w_i ($w_{\min} \le w_i \le 1$) 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 (denoted as r_s and r_e respectively) approximately equal the predefined values. The temperature reduces according to Lundy and Mees' 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.

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 deduction rate $\beta = (t_s - t_e) \cdot nrep/(K \cdot t_s \cdot t_e)$;
Set $t = t_s$; $t_{imp} = t_s$; *iter* = 0; C_a = 0; f_r=false;
Iterative improvement:
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 ++; c_i^{total} ++;
  If ($\delta < 0$) $s = s'$; $t_{imp} = t$; f_r = false; c_i^{accept} ++; c_i^{new} ++; C_a ++;
  If ($\delta > 0$)
    c_i^{new} ++;
    If (exp(-δ/t)<*random*(0,1) $s = s'$; c_i^{accept} ++; C_a ++;
  Endif
  If (δ = 0 && new solution generated) $s = s'$; c_i^{accept} ++; c_i^{new} ++; C_a ++;
  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
  If (mod(*iter*, *LP*)=0)
    If (C_a /*LP* < r_e)
      f_r=true; $t_{imp} = t_{imp}/(1 - \beta t_{imp})$; $t = t_{imp}$; $s = s_{best}$;
      For each i=1,…,n
        If (c_i^{total} =0) $w_i = w_{\min}$;
        Else $w_i = \max\{w_{\min}, c_i^{new}/ c_i^{total}\}$
        Endif
        C_a = 0; c_i^{accept} =0, c_i^{new} =0; c_i^{total} =0;
      Endfor
    Else
      For each i=1,…,n
        If (c_i^{total} =0) $w_i = w_{\min}$;
        Else $w_i = \max\{w_{\min}, c_i^{accept}/ c_i^{total}\}$
        Endif
        C_a = 0; c_i^{accept} =0, c_i^{new} =0; c_i^{total} =0;
      Endfor
    Endif
  Endif
Until *iter* > *K*

Fig. 2. Pseudo-code of the proposed simulated annealing hyper-heuristic (for a minimisation problem).

- Since the performance of a heuristic may change at a different temperature or 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. A few counters are used: c_i^{total} counts the total number of calls of heuristic *i* by the hyper-heuristic during the current learning period; c_i^{new} records how many 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 triggered when the acceptance ratio is getting 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 ($c_i^{accept}/ c_i^{total}$) 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).

In the next section we will demonstrate the high level of generality of the proposed methodology by letting it run on three very different optimisation problems: nurse rostering, course timetabling, and bin packing. The key contribution to the literature is that we have developed a methodology which can operate across 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 close to or even better than special purpose algorithms that have been tailored for just one problem.

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 all three 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% objective-detrimental 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. For the purpose of a fair comparison with other approaches, the total number of repetitions (K) will be changed for different applications. Therefore, we do not need to 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.

4 Computational Experiments

4.1 An application to the nurse rostering problem

In this section, we will apply our simulated annealing hyper-heuristic (SAHH) to nurse rostering. The problem involves producing weekly or monthly schedules for nurses in large hospitals subject to various constraints. In this paper, we address a real nurse rostering problem faced by a large UK hospital. The problem was originally studied in (Dowsland 1998) and later by (Aickelin and Dowsland 2000, Dowsland and Thompson 2000, Aickelin and Dowsland 2003, Aickelin and Li 2006, Aickelin, Burke and Li 2006). The methods described in these papers represent algorithms specifically designed for this domain. In this problem, a feasible schedule has to assign sufficient nurses to cover 14 shifts throughout a week, which can be further divided into 7 day shifts and 7 night shifts. The schedule should be able to cope with the days-off requested by nurses. It should also separate some unpopular shifts (e.g. night shifts and weekend shifts) among nurses for fairness. The problem is particularly challenging because of the need to schedule some of the part-time nurses whose working shifts have to respect their contracts and government regulations. In addition, nurses have different grades. A higher grade nurse can cover the demand for a lower grade nurse but not vice versa. Each nurse can work on one of a number of predefined “shift patterns” with an associated penalty value to represent its preference. For a more detailed discussion of this problem see (Aickelin and Dowsland 2000) and for an overview of nurse rostering in general see (Burke et al. 2004).

The problem was initially addressed by a two-stage tabu search procedure (Dowsland 1998) with good results being obtained. However, this algorithm adopts several types of “chain-moves” which heavily rely upon problem-specific information. Aickelin and Dowsland (2000) used a genetic algorithm in order to reduce the domain-dependency of the tabu search algorithm but their GA is unable to produce high quality results. Further effort was made in one of their more recent publications (Aickelin and Dowsland 2003) which utilised a different solution encoding/decoding scheme and some specialised genetic operators in an indirect genetic algorithm framework. Better results have been obtained by this method. However, these modifications inevitably increased the algorithm’s domain-dependency. Burke, Kendall and Soubeiga (2003) applied a tabu search hyper-heuristic algorithm (TSHH) to this nurse rostering problem. The algorithm used very little domain-specific information but is able to produce better results than the genetic algorithm in (Aickelin and Dowsland 2000) in terms of feasibility. Although the algorithm is not able to produce better solutions than the GA in terms of objective

value, it demonstrates a higher level of generality when dealing with different problem instances such as university course timetabling.

In this paper, we apply our simulated annealing hyper-heuristic to this problem. We only need to have a simple method to produce an initial solution, an evaluation function and the set of low-level heuristics.

4.1.2 Problem model

We are given a number, N, of nurses where each nurse has a grade ranging from 1 to g. Let G_r be the set of nurses with grades r or higher, R_{kr} be the minimal demand of nurses of grade r for shift k and F_i be the set of feasible shift patterns for nurse i. Set $a_{jk} = 1$ if pattern k covers shift j and 0 otherwise. Let p_{ij} be the penalty cost of nurse i working on pattern j. The decision variables x_{ij} take value 1 if nurse i works on pattern j and 0 otherwise. The objective is to minimise the total penalty cost. The problem can be formulated as follows:

$$\min \quad PC = \sum_{i=1}^{N} \sum_{j \in F_i} p_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in F_i} x_{ij} = 1 \tag{2}$$

$$\sum_{i \in G_r} \sum_{j \in F_i} a_{jk} x_{ij} \geq R_{kr} \quad \forall k, r, \tag{3}$$

$$x_{ij} \in \{0,1\} \tag{4}$$

Constraint (2) ensures that each nurse works on exactly one specific shift pattern. Constraint (3) makes sure that there are sufficient nurses to cover each shift at each grade.

4.1.2 Constraint handling

As shown in (Dowsland 1998), the problem is highly constrained and finding a high quality, feasible solution is very difficult. To handle the covering constraints (3), Dowsland (1998) used a search procedure which repeatedly oscillates between feasible and infeasible regions. At each iteration, the algorithm starts from an infeasible solution and uses specialised neighbourhood moves to find a feasible solution (stage 1). Once a feasible solution is identified, the algorithm then attempts to improve it in terms of penalty cost by searching within the feasible region only (stage 2). Both stages used several neighbourhood structures, including chain neighbourhoods, based on a tabu search procedure. In the hyper-heuristic method of (Burke, Kendall and Soubeiga 2003), the covering constraints (3) were handled by using a transformed evaluation function. In this paper, we shall use the same technique. The evaluation function is defined as follows:

$$E = PC + \alpha \cdot Inf \tag{5}$$

where α is a coefficient parameter and *Inf* is a function to measure the degree of infeasibility of a solution, defined as follows:

$$Inf = \sum_{r=1}^{g} (\rho \cdot Bal_r + 1) \sum_{k=1}^{14} \max\{R_{kr} - \sum_{i \in G_r} \sum_{j \in F_i} a_{jk} x_{ij}, 0\} \quad (6)$$

where Bal_r takes value 2 if both day and night are unbalanced at grade r and 1 if either day or night is unbalanced at grade r. ρ is a severity parameter and is set to ρ=5 as given in (Burke, Kendall and Soubeiga 2003).

Coefficient α in the evaluation function (5) is determined based on nurse shortages in the least feasible solution so far, defined by the following function

$$q = \sum_{k=1}^{14} \sum_{r=1}^{g} \max\{R_{kr} - \sum_{i \in G_r} \sum_{j \in F_i} a_{jk} x_{ij}, 0\} \quad (7)$$

and we use $\alpha = 8 \cdot q$ as given in (Burke, Kendall and Soubeiga 2003).

4.1.3 Low-level heuristics

A total of nine low-level heuristics are used. These simple low-level heuristics are the same as those used in (Burke, Kendall and Soubeiga 2003). A description of these heuristics is reproduced here for completeness.

H1. Change the shift-pattern of a random nurse to another random feasible shift pattern;

H2. Similar to H1 except that the acceptance criteria is "1st improving *Inf* value";

H3. Same as H1 but "1st improving *Inf* and not deteriorating *PC*;

H4. Same as H1 but "1st improving *PC*";

H5. Same as H1 but "1st improving *PC* and not deteriorating *Inf*";

H6. Switch the shift-pattern type (i.e. from day shift to night shift and vice versa) of a random nurse if the solution is unbalanced;

H7. This heuristic tries to generate a balanced solution by switching the shift pattern type (i.e. change a day shift-pattern with a night one if night shift(s) is unbalanced and vice versa. If both days and nights are not balanced, swap the shift patterns of two nurses who are working on different shift-pattern types;

H8. This heuristic tries to find the first move that improves *PC* by changing the shift pattern of a random nurse and assign the abandoned shift pattern to another nurse;

H9. Same as H8 but "1st improving *PC* without worsening *Inf*".

4.1.4 Computational results

The above evaluation function and low-level heuristics were input into our simulated annealing hyper-heuristic (SAHH) and the algorithm was tested on 52 problem instances that were used both in (Burke, Kendall and Soubeiga 2003) and (Aickelin and Dowsland 2003). The number of iterations is set to K=12,000 after some preliminary experiments. All the other parameters with regard to SAHH are the same as detailed in section 3. The algorithm was implemented in Microsoft Visual C++ 6.0 and run on a PC Pentium IV 1.8GHZ with 2GB RAM running Microsoft Windows XP Professional. For each instance, the simulated annealing hyper-heuristic started from a random (in almost all cases, infeasible) solution and was run 20 times with different random seeds. Each run took approximately 20 seconds.

We made the following indirect comparison with the tabu search hyper-heuristic (TSHH) of (Burke, Kendall and Soubeiga 2003). In (Burke, Kendall and Soubeiga 2003), it was shown that TSHH is more favourable than the genetic algorithm of (Aickelin and Dowsland 2000) in terms of feasibility but is worse when comparing average objective penalty costs. In the next few paragraphs, we will show that SAHH is superior to an indirect genetic algorithm (Aickelin and Dowsland 2003), an improved version of the genetic algorithm (GA) presented in (Aickelin and Dowsland 2000) both in terms of feasibility and objective penalty costs. This demonstrates that SAHH is also superior to TSHH in terms of objective penalty costs. In addition, we cannot make a direct comparison with results in (Dowsland 1998) since problem instances used in this paper are different from those in (Aickelin and Dowsland 2003) and (Burke, Kendall and Soubeiga 2003).

Table 1. SAHH vs the indirect GA for nurse rostering problems

		Indirect GA*					**SAHH§**			
Instance	**IP**	best	mean	worst	feas%	stdev	best	mean	worst	stdev
1	8	8	8.30	10	100%	0.73	8	8	8	0.00
2	49	51	59.85	66	100%	2.98	49	50.90	55	2.13
3	50	51	60.85	80	100%	8.20	50	50.00	50	0.00
4	17	17	17.00	17	100%	0.00	17	17.00	17	0.00
5	11	11	11.10	13	100%	0.45	11	11.00	11	0.00
6	2	2	2.15	3	100%	0.37	2	2.00	2	0.00
7	11	12	21.95	30	100%	7.55	11	11.00	11	0.00
8	14	15	19.05	35	100%	4.22	14	14.10	15	0.31
9	3	4	5.95	17	100%	4.76	3	3.00	3	0.00
10	2	3	4.10	5	100%	0.72	2	2.50	4	0.69
11	2	2	2.80	5	100%	1.32	2	2.00	2	0.00
12	2	2	2.10	4	100%	0.45	2	2.00	2	0.00
13	2	2	2.30	3	100%	0.47	2	2.00	2	0.00
14	3	3	7.85	15	100%	5.01	3	3.25	4	0.44
15	3	3	4.50	5	100%	0.89	3	3.00	3	0.00
16	37	39	43.95	49	100%	4.19	37	41.40	66	8.66
17	9	10	15.15	37	100%	7.44	9	10.05	15	1.67
18	18	18	19.95	22	100%	0.76	18	18.65	28	2.25

19	1	1	2.30	10	100%	2.08	1	1.25	4	0.72
20	7	7	15.10	23	100%	5.91	7	8.25	21	3.18
21	0	0	0.35	3	100%	0.88	0	0.20	1	0.41
22	25	25	25.65	28	100%	0.75	25	25.30	27	0.57
23	0	0	0.90	2	100%	0.64	0	0.20	1	0.41
24	1	1	1.10	2	100%	0.31	1	1.00	1	0.00
25	0	0	0.00	0	100%	0.00	0	0.20	1	0.41
26	48	48	110.25	200	60%	75.29	48	91.10	198	54.04
27	2	4	13.50	38	100%	10.52	2	4.35	13	4.49
28	63	64	125.55	200	55%	69.10	63	63.25	65	0.55
29	15	15	24.50	200	95%	41.31	15	15.25	18	0.72
30	35	38	41.10	48	100%	2.92	35	35.70	40	1.42
31	62	65	77.15	89	100%	5.44	62	64.70	66	1.84
32	40	42	46.45	52	100%	2.31	40	40.05	41	0.22
33	10	12	14.65	16	100%	1.39	10	15.55	103	20.64
34	38	39	44.20	55	100%	3.61	38	38.05	39	0.22
35	35	36	43.70	53	100%	4.71	35	35.85	39	1.23
36	32	32	36.85	49	100%	5.42	32	32.65	33	0.49
37	5	5	9.05	14	100%	2.44	5	5.00	5	0.00
38	13	15	19.35	29	100%	3.75	13	13.10	15	0.45
39	5	5	7.55	10	100%	1.64	5	5.00	5	0.00
40	7	7	9.90	19	100%	3.74	7	7.45	9	0.69
41	54	55	68.30	86	100%	9.60	54	61.65	83	12.11
42	38	39	43.70	57	100%	6.31	38	38.75	40	0.55
43	22	23	24.25	26	100%	0.85	22	23.20	32	3.04
44	19	25	31.15	40	100%	4.20	19	27.25	34	5.21
45	3	3	4.75	9	100%	1.92	3	3.35	5	0.75
46	3	6	8.10	10	100%	1.07	3	4.80	6	0.83
47	3	3	3.75	6	100%	0.85	3	3.00	3	0.00
48	4	4	8.55	12	100%	2.35	4	4.25	5	0.44
49	27	30	183.40	200	10%	51.11	27	31.85	118	20.28
50	107	110	191.00	200	10%	27.70	107	107.85	109	0.81
51	74	74	87.40	200	90%	38.51	74	74.10	75	0.31
52	58	58	67.35	85	100%	8.36	58	59.30	73	3.92

[*] An arbitrary penalty of 200 is used for the infeasible solutions by the indirect GA.
[§] SAHH is able to create feasible solutions for all 20 runs of every problem instance, i.e. feas%=100%. For space reasons, this information is not included in this table.

Table 1 presents the results produced by SAHH and the indirect genetic algorithm (Indirect GA) (Aickelin and Dowsland 2003) (we did not make comparisons with results in (Dowsland 1998) since the problem instances in (Dowsland 1998) are different from those in (Aickelin and Dowsland 2003)). Both the best, average and worst results and standard deviations across 20 runs are reported. All the instances have been solved to optimality by an integer programming procedure (IP) (Dowsland and Thompson 2000). However, the computational time is extremely high for some instances. Fig. 3 (respectively Fig. 4) plots more detailed results by the indirect GA (respectively SAHH), including the number of infeasible solutions (#inf), the number of optimal solutions found (#opt) and the number of solutions being within 3 cost units away from the optimum (i.e. good quality solutions) among 20 runs.

It can be seen that SAHH produced preferable results than the indirect GA in terms of feasibility as well as the objective penalty costs. The indirect GA struggles to find feasible solutions for a few instances, especially for instance 49 and 50 where the feasibility ratio is only 10%. However, like TSHH, SAHH can produce feasible solutions for all 20 runs of every problem instance. In terms of objective penalty cost, SAHH produced better quality solutions than the indirect GA for the majority of the problem instances. For 15 instances, SAHH is able to consistently find optimal solutions (i.e. #opt=20) while the indirect GA only managed 2 instances. Among 20 runs, SAHH is able to solve all instances to optimality, compared to 54% (28 out of 52) by the indirect GA. When compared with the average computational results (see table 2), it can be seen that SAHH is better than the indirect GA for 49 instances and has the same level of performance for 1 instance. The indirect GA has slightly better average results than SAHH for the remaining 2 instances.

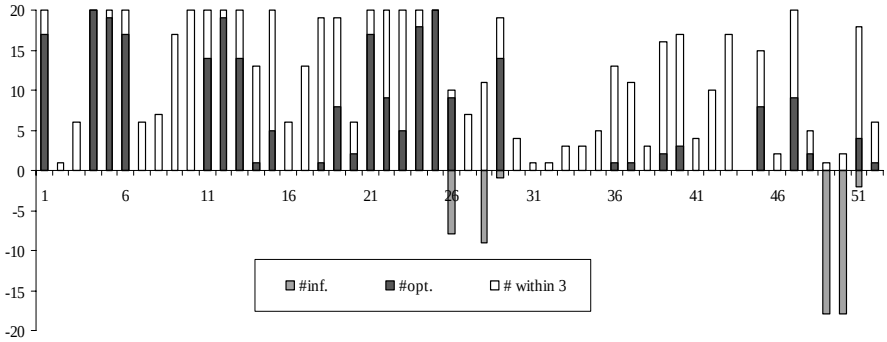

Fig. 3. Detailed results of the indirect GA for nurse rostering problems among 20 runs

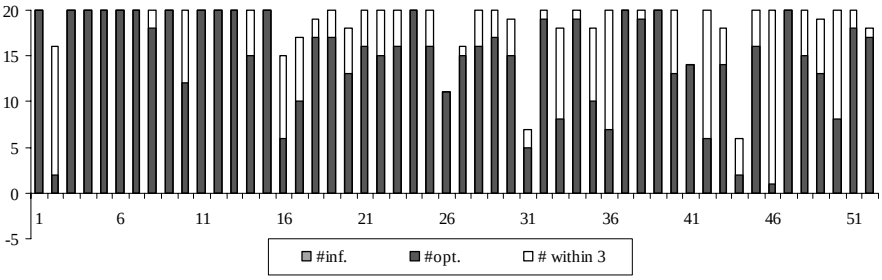

Fig. 4. Detailed results of SAHH for nurse rostering problems among 20 runs

Fig. 5 shows the standard deviation of the indirect GA and SAHH over 20 runs. It can be seen that SAHH seems to be more consistent than the indirect GA. In the majority of cases, the standard deviation by SAHH is much lower than the indirect GA.

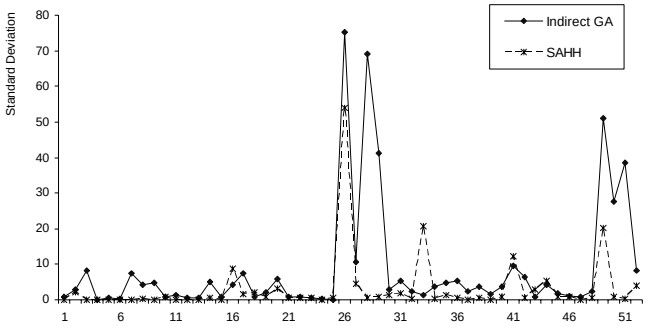

Fig. 5. Standard deviation of indirect GA and SAHH for nurse rostering problem

From the above comparisons, it can be concluded that SAHH is able to produce high quality results which are better than those of the indirect GA for these nurse rostering problem instances, both in terms of feasibility rates and objective penalty costs. The key point here is that a method (SAHH) which has been designed to be generic across different problems is able to outperform recently published methods that were designed to solve only this nurse rostering problem. TSHH was designed to be robust across different problems but, since the indirect GA can produce superior results to TSHH, we can therefore conclude that SAHH is also superior to TSHH for this problem. In addition, the results by SAHH are also better than the recently results in (Aickelin and Li 2006, Aickelin, Burke and Li 2006).

4.2 An application to university course timetabling

4.2.1 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. It is a very different problem to the nurse rostering problem discussed above. For the purpose of comparison, we shall use the same model presented in (Rossi-Doria et al. 2002) and (Socha, Knowles and Samples 2002). This model provides a representation of a typical course timetabling problem and has been widely used in recent publications. The model formulates the problem as follows:

Given a set of events e_i (i = 0,…,E) and a number of rooms r_j (j = 0,…,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,…,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, Knowles and Samples 2002) and (Burke, Kendall and Soubeiga 2003). 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, Knowles and Samples 2002) and (Burke, Kendall and Soubeiga 2003), room assignments are dealt with separately by using a matching algorithm.

4.2.2 Low-level heuristics

The three simple low-level heuristics that we use can be outlined 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, Burke and McCollum 2005).

4.2.3 Computational results

The proposed simulated annealing hyper-heuristic algorithm (SAHH) was tested on two course timetabling benchmark data sets. The first data set was originally used in (Socha, Knowles and Samples 2002). It consists of five small instances (E=100, S=80, R=5 F=5), five medium instances (E=400, S=200, R=10 F=5) and one large instance (E=400, S=400, R=10 F=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: $E \in$ [350,440], $S \in$ [200,350], $R \in$ [10,11], $F \in$ [5,10]. 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, Burke and Garibaldi 2005). However, to quickly obtain a feasible initial solution that 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 total 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 one minute computational time for small instances and one and half minutes for medium and large instances, running on the same machine as for the nurse rostering problem described in section 4.1.4. Furthermore, since computational time is generally considered to be non-critical for the course timetabling problem, a larger number of iterations (K=2*10^7) were also investigated for this data set. For the second data set (the competition data set), we set K=2*10^7 which corresponds to approximately between 45 and 55 minutes computational time on the same machine. All the other parameters remain unchanged. In a similar way to that undertaken by Burke, Kendall and Soubeiga (2003), five independent runs were carried out for each problem instance using different random seeds and both best results 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, Kendall and Soubeiga 2003), 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 2. A comparison of SAHH with other approaches for the course timetabling problem on the first data set

	ANT	**VNS**	**TSHH**		**GHH**	**SAHH** (K=500000)		**SAHH** (K=2*10^7)	
Datasets	mean	best	best	mean	best	best	mean	best	mean
small1	1	**0**	1	2.2	6	**0**	0.6	**0**	**0.0**
small2	3	**0**	2	3.0	7	**0**	2.2	**0**	**0.0**
small3	1	**0**	**0**	1.4	3	1	1.2	**0**	**0.0**
small4	1	**0**	1	1.8	3	1	1.8	**0**	0.2
small5	**0**	**0**	**0**	0.2	4	**0**	0.6	**0**	**0.0**
medium1	195	317	146	179.0	372	102	117.0	**38**	**45.8**
medium2	184	313	173	197.6	419	114	122.0	**28**	**36.2**
medium3	248	357	267	295.4	359	125	150.2	**48**	**54.2**
medium4	164.5	247	169	180.0	348	106	110.6	**21**	**27.0**
medium5	219.5	292	303	388.5	171	106	143.2	**12**	**18.2**
large	851.5	N/A	1166	1166.0	1068	653	670.2	**519**	**569.2**

Table 2 presents a comparison between SAHH and other meta-heuristic and hyper-heuristic approaches reported in the literature for the first data set instances, including an ant algorithm (ANT) (Socha, Knowles and Samples 2002), tabu search hyper-heuristic (TSHH) (Burke, Kendall and Soubeiga 2003), variable neighbourhood search (VNS) (Abdullah, Burke and McCollum 2005) and a graph-based hyper-heuristic (GHH) (Burke et al. 2007). Both best and average results are reported. The best results are presented in bold. It can be

seen that across all eleven problem instances, the proposed simulated annealing hyper-heuristic not only outperformed the other two hyper-heuristics (TSHH and GHH), but it also produced much better quality solutions when compared with two problem-tailored meta-heuristics (ANT and VNS).

When given a relatively small computational time (i.e. K=500000), SAHH can produce competitive results for all eleven problem instances, compared with the other approaches. For the five small instances, SAHH is able to solve three out of five instances to optimality within five runs. For the remaining two instances, the solutions obtained by SAHH are very close to optimality. For the five medium instances and the large instance, SAHH has shown significant improvement over the other meta-heuristic and hyper-heuristic approaches. Both the best results and average results obtained by SAHH are much better than the current best known solutions for these problems. When comparing the best results by SAHH with TSHH, SAHH reduced the objective value by between 30% and 65% with an average reduction of 44%. Comparing the average results by SAHH with the median results by ANT, the percentage of reduction in objective values ranges from 20% to 53% with an average reduction of 37%. The reduction rates are even higher when compared with VNS and GHH. It seems that the proposed simulated annealing hyper-heuristic is particularly suited to instances with larger search spaces. This is probably due 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. 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**

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 (Metaheuristics Network 2003). 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 due to 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 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 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 the perfect 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.

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

Across both 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 seen in section 4.1 and the next section, all the parameters (except the total number of iterations) used in the simulated annealing hyper-heuristic are the same for all problems and problem instances.

4.3 An application to bin packing

In the previous two sections, 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 on two search problems. In fact, it also produced better results than several different 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 experiments. As before, we only need to change the initial solution, the evaluation function and the set of low-level heuristics.

4.3.1 The bin packing problem

The one dimensional bin packing problem is defined as follows. Given a set of items I (i =1,…, 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, Klein and Jurgens 1997). Considerable research has been carried out on both exact methods and heuristic based approaches. 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, Klein and Jurgens 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 1998). Recently, Alvim et al. (Alvim et al. 2004) proposed a hybrid algorithm which combines several strategies. 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. Ross et al. (2003) proposed a genetic algorithm hyper-heuristic for the problem. The algorithm was firstly trained on some benchmark problems and after the training, the fittest

chromosome was then applied to every benchmark problem. However, only 80% of the problem instances were solved to optimality.

4.3.2 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 average slack value allowed in the lower bound L_1) and 0.02 seconds 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, Klein and Jurgens 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 shift them 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 seconds 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.

4.3.3 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 1000 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 1000 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), 1000 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 1210 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 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/adriana/_les/detbp.pdf).

4.3.4 Computational results

The above low-level heuristics and initial solutions were input to our simulated annealing hyper-heuristic framework which, again, was run on a PC Pentium IV 1.8GHZ with 256MB RAM running Windows XP Professional. The algorithms were run 10 times for each instance, using a different random seed each time. For each problem instance, the simulated annealing hyper-heuristic stops either after K = 200,000 iterations or when 20 seconds of computational time is exceeded. 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 10 runs were reported. For 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).

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	0.04	0.61
Fal_U250	20	**19**	1	0.03	0.16	**19.0**	1	1.23	20.02
Fal_U500	20	**20**	0	0.04	0.14	**20.0**	0	1.31	20.02
Fal_U1000	20	**20**	0	0.07	0.27	**20.0**	0	0.57	7.70
Fal_T60	20	**20**	0	0.01	0.01	**20.0**	0	0.93	3.69
Fal_T120	20	**20**	0	0.02	0.04	**20.0**	0	1.11	5.17
Fal_T249	20	**20**	0	0.02	0.04	**20.0**	0	0.56	2.57
Fal_T501	20	**20**	0	0.06	0.10	**20.0**	0	0.95	2.84
Sch_Set1	720	694	2	0.15	1.78	**701.7**	1	0.43	20.02
Sch_Set2	480	474	1	0.10	4.57	**476.7**	1	0.36	20.42
Sch_Set3	10	2	1	3.74	5.05	**9.1**	1	5.44	20.11
All	1370	1329	2	0.14	5.05	**1346.5**	1	0.49	20.42

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	0.04	0.61
Fal_U250	20	**20.0**	0	0.15	3.19	19.0	1	1.23	20.02
Fal_U500	20	**20.0**	0	0.00	0.01	**20.0**	0	1.31	20.02
Fal_U1000	20	**20.0**	0	0.01	0.03	**20.0**	0	0.57	7.70
Fal_T60	20	**20.0**	0	0.33	2.53	**20.0**	0	0.93	3.69
Fal_T120	20	**20.0**	0	1.14	6.88	**20.0**	0	1.11	5.17
Fal_T249	20	**20.0**	0	0.29	2.91	**20.0**	0	0.56	2.57
Fal_T501	20	**20.0**	0	1.24	19.26	**20.0**	0	0.95	2.84
Sch_Set1	720	**719.2**	1	0.20	23.89	701.7	1	0.43	20.02
Sch_Set2	480	**480.0**	0	0.01	1.89	476.7	1	0.36	20.42
Sch_Set3	10	**10.0**	0	4.71	50.61	9.1	1	5.44	20.11
Sch_Wae1	100	**100.0**	0	0.02	0.14	99.1	1	0.65	20.04
Sch_Wae2	100	**100.0**	0	0.02	3.58	99.1	1	0.39	20.17
Wae_Gau1	17	**12.0**	1	0.60	2.40	**12.0**	1	5.90	20.01
All	1587	**1581.2**	1	0.38	50.61	1556.7	1	0.55	20.42

Tables 5 and 6 respectively present a comparison of our simulated annealing hyper-heuristic with a hybrid variable neighbourhood search algorithm (Perturbation MBS' + VNS) (Fleszar and Hindi 2002) and a recently proposed hybrid procedure (HP_BP) in (Alvim et al. 2004) (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). It can be seen that our simulated annealing hyper-heuristic produced similar results to the hybrid VNS in the first group of data sets and slightly better results in 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 17 more instances than the hybrid VNS. In terms of computational time, our simulated annealing