A Tabu-Search Hyperheuristic for Timetabling and Rostering

E.K. Burke, G. Kendall, and E. Soubeiga*
School of Computer Science & IT, University of Nottingham
Nottingham NG8 1BB, UK; ekb|gxk|exs@cs.nott.ac.uk

Abstract

Hyperheuristics can be defined to be heuristics which choose between heuristics in order to solve a given optimisation problem. The main motivation behind the development of such approaches is the goal of developing automated scheduling methods which are not restricted to one problem. In this paper we report the investigation of a hyperheuristic approach and evaluate it on various instances of two distinct timetabling and rostering problems. In the framework of our hyperheuristic approach, heuristics compete using rules based on the principles of reinforcement learning. A tabu list of heuristics is also maintained which prevents certain heuristics from being chosen at certain times during the search. We demonstrate that this tabu-search hyperheuristic is an easily re-usable method which can produce solutions of at least acceptable quality across a variety of problems and instances. In effect the proposed method is capable of producing solutions that are competitive with those obtained using state-of-the-art problem-specific techniques for the problems studied here, but is fundamentally more general than those techniques.

Key words: Hyperheuristic, Tabu Search, Heuristic, Scheduling, Rostering, Timetabling, Local search.

1 Introduction

The past twenty years or so have seen the development of a number of metaheuristics, including tabu search, simulated annealing, ant algorithms and genetic algorithms (e.g. see [17, 29, 30, 38]). In order to generate an automated method for a given NP-hard optimisation problem, a software developer must produce a solution technique which takes into account the context within which the technique will be used (e.g. does the problem owner place much emphasis on solution quality, is the problem owner prepared to invest a lot of time in the development of the technique, is computational expense an issue, etc). In order to determine which solution approach to use it is important to distinguish between the following two scenarios.

- *Scenario a*: We are looking for the best solutions possible and preferably one that is optimal. Any slight improvement on the solutions will make a big difference. We are prepared to invest time and money in order to obtain these solutions.
- *Scenario b*: This is not a one-off problem but an instance of a series (class) of problems which we encounter regularly. We want to use the same method for a number of other problems without having to spend a lot of time re-adapting the method to each new instance or new problem tackled. We would much prefer one method as long as the solutions produced are of acceptable quality across the various problems which we have to solve. One 'general' method is far more efficient to implement (and market) than a range of 'problem-specific' approaches. We require solutions which are 'good enough - soon enough - cheap enough' [5].

In the first scenario, it may be worth considering exact algorithms, or the development of bespoke metaheuristic techniques, which, in practice, are often hybridised with problem specific heuristics. Sometimes

*Corresponding author

these heuristics are even specific to particular problem instances. Metaheuristics have been proven to be very successful for solving various optimisation problems in such a context [17, 27, 29, 30, 38], especially when domain-specific knowledge is incorporated. The advantage of metaheuristics in this scenario is that, once the technique has been developed, good-quality solutions can be produced quite quickly. The drawback of metaheuristics in this scenario, however, is that in order to apply the same technique to a different problem or problem instance, one needs to carry out (sometimes extensive) tuning of the method. This may involve the adjustment of relevant parameters of the technique for the new problem. It may also involve the incorporation of different problem-specific heuristics. For example it was noted in [1, 2] that a tabu search metaheuristic used in [13] relied heavily on some of the problem-specific information such as the coefficients of the objective function. As a result the tabu search of [13] was no longer effective when perturbations of the problem data were made [1]. Some metaheuristics can be brittle, precisely because of their bespoke, tailor-made nature. Hence problem-specific metaheuristics are often not readily applicable to a new problem [9]. Another illustration can be found in [3, 4] where a metaheuristic approach is used for a nurse rostering system. The system produces excellent results for nurse rostering and is in use in over 40 Belgian hospitals. The methods would need significant programming effort though to use them for nurse rostering in different countries because the system is tailor-made for the rules, regulations and working practices that apply in Belgium. If those rules, regulations and working practices change then the system will need altering. This situation is highly appropriate in circumstances where high quality schedules that satisfy a wide range of constraints are crucial (such as the one described above). The problem is that such systems are expensive to implement and often more expensive to adapt to new problems or problem instances. There will always be a place for expensive problem-specific systems which can produce very high quality solutions to specific critical problems. However, there is also a place for much more general systems which can deal with a wide range of problems. In many optimisation scenarios we are not looking for very high quality solutions. Instead, we may well require solutions which are 'good enough - soon enough - cheap enough'.

This paper is concerned with developing the heuristic infrastructure which will allow us to operate efficiently and effectively within this more general framework. The overall goal of such research is to develop systems that can operate at a higher level of generality than today's systems, but which can also produce solutions which are competitive with state-of-the-art problem-specific systems.

In *scenario b* we are looking for a technique which can be easily applied to different problems. This means that we require a method which was not designed with one particular problem (or problem instance) in mind but is instead applicable to a class of problems or domains. Note that the aim is not to solve all problems with one method, but a subset of problems. The No Free Lunch Theorem [39] implies that the former objective cannot be achieved anyway. The aim is to *raise* the level of generality. The method should not require customisation when applied to a new problem, perhaps at the expense of reduced but still acceptable solution quality (when compared with a made-to-measure metaheuristic technique). While the focus in *scenario a* is on solution quality, the focus in *scenario b* is on technique development time or solution-method quality. It is the context of *scenario b* that has motivated the use of hyperheuristic techniques [9]. A hyperheuristic can be defined to be a heuristic which chooses between heuristics. While a metaheuristic usually deals with solutions, a hyperheuristic deals with *solution methods* (e.g. heuristics). A metaheuristic can and usually does modify solutions directly. A hyperheuristic can only modify solutions indirectly, by way of an operator (a low-level heuristic). This places a hyperheuristic at a higher level of abstraction than most current studies of metaheuristics. Of course, hyperheuristics can be metaheuristics. Indeed they usually are [5]. The point of using the term hyperheuristics (rather than metaheuristics) is that it tells us that we are working on trying to find the right method or heuristic in a particular situation rather than trying to solve a problem directly.

The work reported in this paper was conducted in this context. The aim is not to develop a method which would 'beat' existing algorithms for a given optimisation problem, but instead a method which is capable of performing well-enough, soon-enough, cheap-enough across a range of problems and domains. Such a method could ultimately underpin cheaper optimisation systems which would be available to a wider range of users than is the situation today. A hyperheuristic can choose which low-level heuristic to apply at each decision point, until a stopping condition has been fulfilled. In order to apply a hyperheuristic to a

given optimisation problem, we only need a set of low-level heuristics and an evaluation function to assess solution quality. A key ingredient in implementing a hyperheuristic is the learning mechanism, which guides the hyperheuristic in the way in which low-level heuristics are selected.

Over the past few years a number of hyperheuristic studies have been reported in the literature although the term 'hyperheuristic' is not always employed [5, 9, 10, 11, 14, 26, 33, 35, 37]. In fact the term 'hyperheuristic' was introduced relatively recently. See [5] for an overview of hyperheuristic approaches. The first papers which discussed hyperheuristic ideas were published in the early 1960's by Fisher and Thompson [15, 16] and by Crowston, Glover, Thompson and Trawick [12]. All three papers [12, 15, 16] tackled the problem of job-shop scheduling. The learning method was based on probabilistic weighting of the low-level heuristics, which represented various scheduling rules. Subsequent work on hyperheuristics was carried out by Mockus et al. who advocated a Bayesian mechanism for searching between low-level heuristics [19, 20, 23]. More recently Hart and Ross [18] used a genetic algorithm (GA)-based hyperheuristic to solve the job-shop scheduling problem. In their approach the GA chromosome represents which method to use in order to identify conflicts among schedulable operations and which heuristic to use in order to select an operation from the conflicting set. Computational results showed that evolving solution methods was beneficial and results obtained were promising when compared to those most recently published. Another GA-based hyperheuristic approach was that presented by Cowling et al. [8] for solving a trainer scheduling problem. In [8] the GA chromosome represents an ordered sequence of low-level heuristics to be applied to the problem. Results produced by their hyperheuristic were superior to those obtained by both a conventional GA using a direct representation of the problem and a memetic algorithm which invokes a local search improvement procedure after the application of crossover and mutation operators. Nareyek [24, 25] also proposed a hyperheuristic method which uses ideas based on reinforcement learning [21, 36] in order to choose which heuristic to apply at each decision point. In his approach each heuristic is assigned a weight which can increase or decrease according to the heuristic's performance. Various reward and punishment schemes were considered when selecting a low-level heuristic. These reward and punishment schemes were then compared to one another by way of solving two optimisation problems. Another account of a hyperheuristic approach is reported in [31] for solving bin-packing problems. The learning mechanism here is a learning classifier system, in which each classifier represents a bin-packing heuristic or rule. The aim is to use learning classifier systems in order to produce good combinations of bin-packing heuristics and rules. The resulting hyperheuristic method was able to produce bin-packing solutions which were better than the ones obtained when each bin-packing heuristic was used on its own. Cowling et al. [9, 10, 11, 22] report the application of another type of hyperheuristic which uses a choice function to rank the low-level heuristics. The choice function hyperheuristic has been successfully applied to various real-world personnel scheduling problems.

In this paper, we carry out a thorough investigation of a tabu-search hyperheuristic technique. Under this framework, the heuristics compete against one another in order to be selected. We demonstrate the generality of our hyperheuristic by evaluating it on a comprehensive suite of problem instances of nurse-scheduling and university course-timetabling problems. In effect, the proposed method is capable of producing solutions which are competitive with those obtained using the state-of-the-art techniques for both problems.

The remainder of this paper is organised as follows. In section 2 we give a description of the proposed hyperheuristic approach. Computational experiments are then reported for the nurse scheduling problem and the university course timetabling problems in sections 3 and 4 respectively. Section 5 presents our conclusions.

2 A tabu-search hyperheuristic

As previously mentioned, in order to apply a hyperheuristic to a given problem, we only need to input the low-level heuristics and the evaluation function. The hyperheuristic described in this paper can be thought of as being a black box which returns (outputs) a solution to a given problem, by employing a given set of low-level heuristics for the problem. This is illustrated in figure 1. The hyperheuristic black box receives, as input, information about the problem as well as a number of low-level heuristics. It then

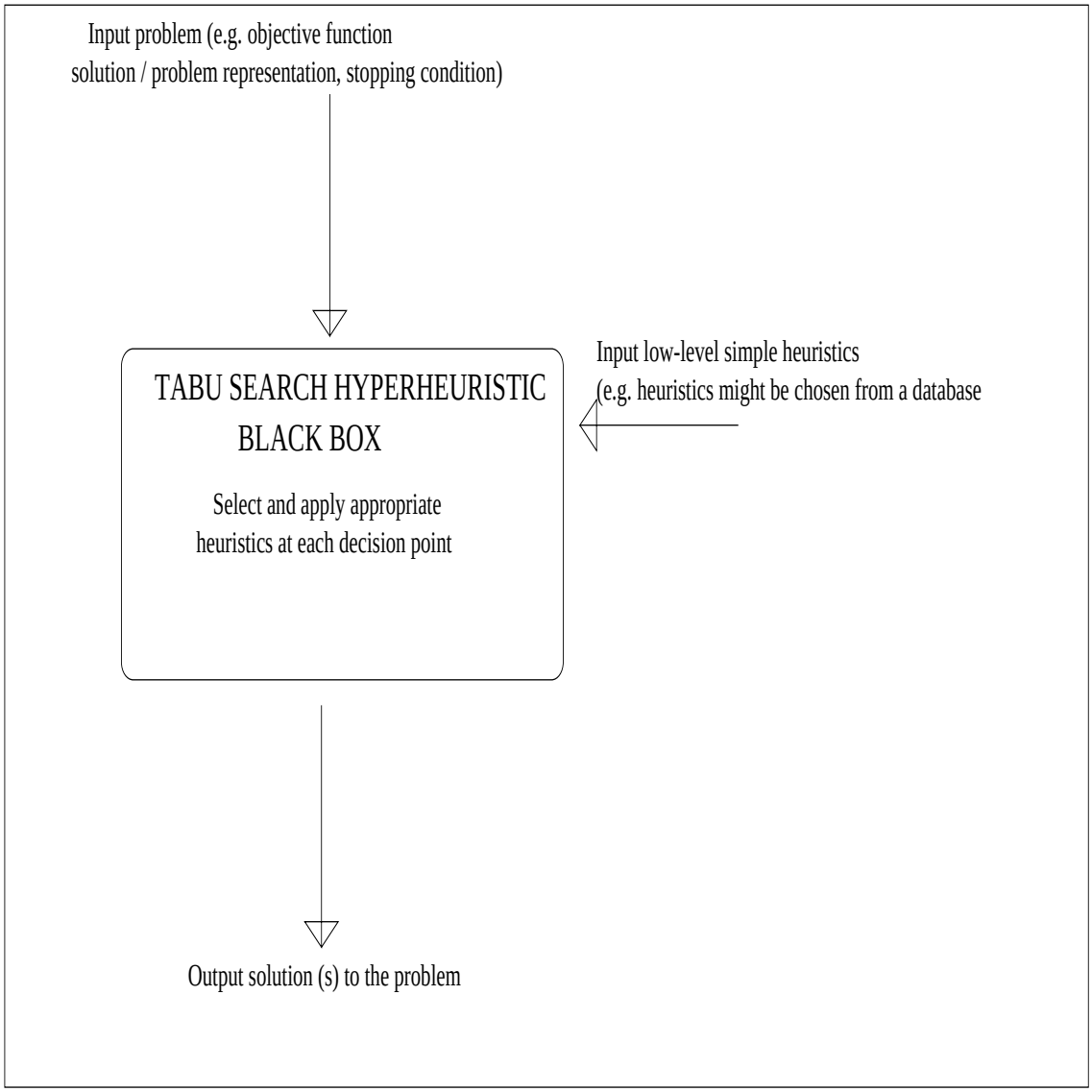


Figure 1: General hyperheuristic framework

selects and applies a heuristic at each decision point. Then, when the stopping condition has been met, the hyperheuristic black box returns, as output, one or several solution(s) to the problem. A detailed description of how this process is carried out is presented in the next paragraph and is illustrated in figure 2. Simple low-level heuristics (e.g. 'add', 'drop', 'swap' objects) can be easily implemented and input to the black box. There is a potential drawback due to the argument that this approach will not underpin a wider take up of more general systems because a user has to input low-level heuristics, which requires specialist knowledge. However, this argument can easily be countered by developing systems which link to a maintained database of low-level heuristics, which could be dragged-and-dropped into the hyperheuristic. Each heuristic in the database could be described in simple natural language terms so that the user can understand what they do. The idea of storing previously built heuristics in a database is not new. The machine learning paradigm of case-based reasoning has recently been used to select heuristics for course timetabling problems [6, 28]. In these two papers, the case-based reasoning system maintains a case base of information regarding the performance of different heuristics on previously solved timetabling problems.

Different types of hyperheuristics will have different ways of selecting the low-level heuristics, e.g. choice function hyperheuristics [9, 10, 11, 22], genetic-algorithm hyperheuristics [8, 18], etc. In this paper we investigate and develop a tabu-search hyperheuristic. In figure 2, we present the framework of our tabu-search hyperheuristic. Both the problem and a number of low-level heuristics have already been input to the hyperheuristic black box. The output of the hyperheuristic is a (set of) solution(s) to the problem. As mentioned earlier, the low-level heuristics can be thought of as competing with each other. The competition rules are inspired from the principles of reinforcement learning [21, 36]. At the beginning of the search each low-level heuristic k has a score, $r_k = 0$, of 0 points. Heuristic ranks are allowed to decrease and increase within the interval $[r_{min}, r_{max}]$, where r_{min}, r_{max} are respectively lower and upper ranks. When a heuristic has been applied we note the change Δ in the evaluation function value from the previous solution to the new one (Δ represents the improvement in the objective function). If this results in an improvement (i.e. $\Delta > 0$) then the score of the heuristic is increased, e.g. $r_k = r_k + \alpha$. Otherwise it is decreased, e.g. $r_k = r_k - \alpha$, where α is a positive number. There are several ways in which α can be chosen. For example α can be selected to be a simple integer or real number. It can also be determined as a function of various parameters such as the change Δ in the objective function value, the amount of CPU time used to apply the chosen low-level heuristic, the individual and collective performance of a set of heuristics, etc. [9, 10, 11, 22]. Here we choose a simple scheme for α; it is taken to be a positive integer number.

In addition, a tabu list of low-level heuristics is maintained, which excludes some heuristics from the competition for a certain duration. Note that while in a standard tabu search metaheuristic approach (which operates directly on a problem), solutions or solution attributes are made tabu, in our tabu-search hyperheuristic approach it is rather *solution methods* (heuristics) that are made tabu. The basic idea of the tabu list is to prevent a heuristic which did not perform well from being chosen too soon (even if it has the highest rank). More precisely we include heuristic k in the tabu list (on a 'First In - First Out' basis) if Δ is non-positive. In addition, heuristics already in the tabu list are released if $\Delta \neq 0$. The idea is that there is no point in keeping a heuristic tabu once the current solution has been modified (e.g. when $\Delta \neq 0$). Thus we employ a variable-length dynamic tabu list of heuristics [7].

Overall the tabu search hyperheuristic can be outlined by the following pseudocode:

Do

1 -Select the low-level heuristic, k, with highest rank and apply it.

2.1 - If $\Delta > 0$ *then* $r_k = r_k + \alpha$

2.2 - Else $r_k = r_k - \alpha$ *and include k in TABULIST*

Until Stopping condition is met.

We use $\alpha = 1$, $r_{min} = 0$ and $r_{max} = n$ as defined in [7] where n is the number of low-level heuristics available. In the next two sections we report results that are obtained by applying the tabu-search hyperheuristic to two different optimisation problems. Again, in order to apply a hyperheuristic to a given problem, all that is needed is a set of low-level heuristics and a means of evaluating solution quality. These will be given for

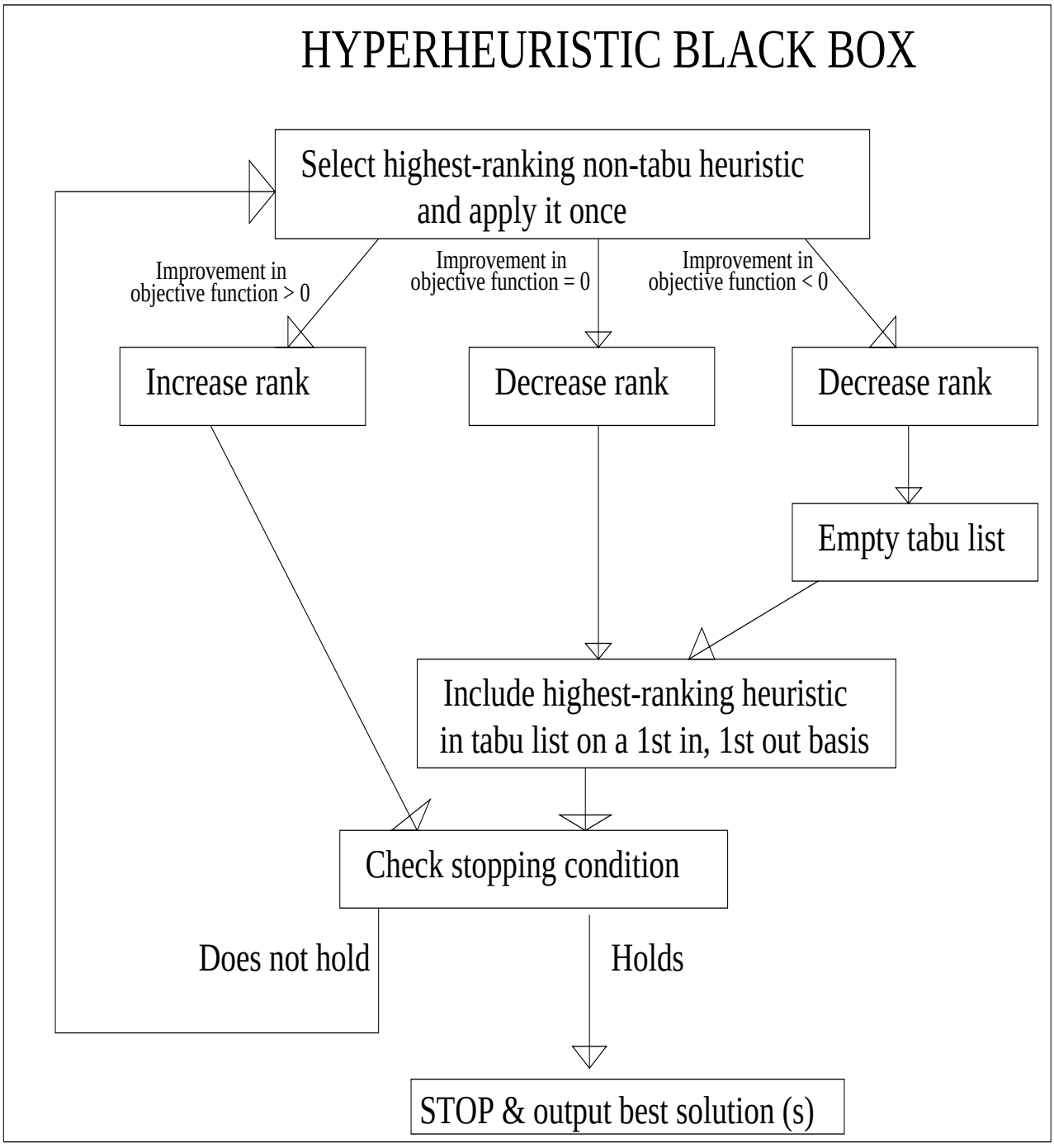


Figure 2: Tabu-search hyperheuristic framework

each problem considered below. Note that the key point of this paper is that the tabu-search hyperheuristic is not designed with a particular problem in mind. On the contrary the goal is to develop an approach which is more general than the current state of the art. It is this generality that is demonstrated in the next two sections.

3 An application to nurse scheduling

3.1 Problem description

This problem is concerned with creating weekly schedules for wards of up to 30 nurses at a major UK hospital. The schedules created must respect working contracts and regulations and meet the demand expressed as the number of nurses of different grades required for each day and night of the week, whilst being perceived to be fair by the nurses themselves. In any given week, nurses work either days or nights. The day is further divided into two types of shift: 'earlies' and 'lates'. A full week's work typically includes more days than nights. This leads to a number of different weekly shift-patterns corresponding to different combinations of days-off and days-on for each week and each nurse. The problem is to allocate a weekly shift-pattern to each nurse. There are typically between 20 and 30 nurses per ward, 3 grade-bands (qualification levels), and 411 different full-time and part-time shift-patterns. Based upon the nurses' individual preferences for the various shift-patterns, the recent history of shift-patterns worked, and the overall attractiveness of the shift-pattern, a penalty cost is determined. This penalty cost is associated with each allocation of a nurse to a shift pattern. Penalty costs were set after close discussion with the hospital, ranging from 0 (ideal) to 100 (undesirable). This problem is described in detail in [13]. Let

- x_{ij} be the decision variables which take value 1 if nurse i works shift-pattern j and 0 otherwise;
- parameter g be the number of grades;
- parameter n be the number of nurses;
- parameter s be the number of possible shift-patterns;
- a_{jk} be a parameter which assumes 1 if shift-pattern j covers shift k, 0 otherwise;
- b_{ir} be a parameter which assumes 1 if nurse i is of grade r or higher, 0 otherwise;
- p_{ij} be the penalty cost of nurse i working shift-pattern j;
- S_{kr} be the demand of nurses of grade r or above on shift k;
- $F(i)$ be the set of feasible shift-patterns for nurse i.

The aim of the problem is to minimise the following penalty cost function:

$$\sum_{i=1}^{n}\sum_{j=1}^{s} p_{ij}x_{ij} \tag{1}$$

subject to the following constraints:

$$\sum_{j\in\mathbf{F(i)}} x_{ij} = 1, \quad \forall i \tag{2}$$

$$\sum_{j=1}^{s}\sum_{i=1}^{n} b_{ir}a_{jk}x_{ij} \geq S_{kr}, \forall k, r \tag{3}$$

The objective, expressed in (1), is to minimise the overall penalty cost associated with the nurses' view of the shift-patterns they have been allocated to. The constraint in (2) enforces each nurse to work exactly

one shift-pattern. Constraint (3) requires that a minimum number of nurses be scheduled for each shift and for each grade-band. Note that b_{ir} is defined in such a way that higher-grade nurses can be substituted for those with lower grades if necessary. The problem is NP-hard [2] and instances typically involve between 1000 and 2000 variables and up to 70 constraints. It was noted in [2] that the difficulty of a given instance is related to the shape of the solution space, which itself depends on the distribution of the penalty costs (p_{ij}) and their relationship with the set of feasible solutions. Here we propose to compare the hyperheuristic results with those published in the literature for 52 instances of the problem, based on three wards and corresponding to each week of the year. This represents a fairly comprehensive suite of problems which feature a wide variety of solution landscapes ranging from easy problems with many low-cost global optima scattered all over the space, to very hard ones with few global optima and in some cases with relatively sparse feasible solutions [2]. The most successful method to date which works within the low CPU time available in practice is a tabu search metaheuristic [13] which uses chain-moves whose design and implementation were very problem-specific. Effectively these chain-moves relied on the way the different factors affecting the quality of a schedule were combined in the p_{ij} penalty costs [2]. In an attempt to reduce the problem-specificity and domain-dependency of the tabu search approach of [13], a genetic algorithm which did not use the chain-moves was developed in [2] but failed to obtain good solutions. This led to the use of a co-evolutionary strategy which decomposed the main population into several co-operative sub-populations. However, domain-knowledge of the problem structure was incorporated in both the way the sub-populations were built, and the way partial solutions were recombined to form complete ones. The co-evolutionary strategy of [2] is very much problem-specific, and, both the tabu search of [13] and the genetic algorithm of [2] are only applicable to problems with a very similar structure.

In this paper, we solve this nurse scheduling problem using our tabu-search hyperheuristic which does not rely upon the problem-specific information required by the tailor-made approaches of [2] and [13]. We use the hyperheuristic evaluation function of [11, 7]. Given that nurses work either days or nights it is the case that in order for a given solution to be feasible, (i.e. enough nurses covering all 14 shifts at each grade-brand), the solution must have sufficient nurses in both days and nights independently of each other. Recall that, in a given week, nurses work either days or nights, but not both. A solution is defined as balanced in days (or nights) at a given grade-band r if there are both under-covered and over-covered shifts in the set of days (or nights) at grade r such that the nurse surplus in the over-covered day (or night) shifts suffices to compensate for the nurse shortage of the under-covered day (or night) shifts. A solution cannot be made feasible until it is balanced [13, 2]. The degree of infeasibility of a solution as defined in [11] is

$$Infeas = \sum_{r=1}^{g}(\rho \times Bal_r + 1) \sum_{k=1}^{14} max\left(\left[S_{kr} - \sum_{i=1}^{n}\sum_{j=1}^{s} b_{ir}a_{jk}x_{ij}\right], 0\right),$$

where Bal_r is 2 if both day and night are unbalanced at grade r, 1 if either day or night is unbalanced at grade r, and 0 otherwise; ρ is a severity parameter for unbalanced solutions, whose value is chosen so that a balanced solution with more nurse-shortages is preferred to an unbalanced one with fewer nurse-shortages, as the latter is more difficult to make feasible than the former. We use $\rho = 5$ as given in [11]. The evaluation function can then be defined as

$$E = PC + C_{demand}InFeas$$

with C_{demand} a weight associated with $InFeas$ as in [2]. The definition of C_{demand} is based on the number, q, of nurse-shortages in the best least-infeasible solution so far, i.e.

$$q = \sum_{k=1}^{14}\sum_{r=1}^{g} max\left(\left[S_{kr} - \sum_{i=1}^{n}\sum_{j=1}^{s} b_{ir}a_{jk}x_{ij}\right], 0\right).$$

Coefficient C_{demand} of $InFeas$ in E is then given by $C_{demand} = \gamma \times q$ if $q > 0$, and $C_{demand} = v$ otherwise; where γ is a preset severity parameter, and v is a suitably small value. The idea is that the weight C_{demand} depends on the degree of infeasibility in the best least-infeasible solution encountered so far, after which it remains at v. We use $\gamma = 8$ and $v = 5$ as given in [2] so as to have to make a fair comparison. Note

that we weight feasibility more highly than we weight cost.

3.2 Low-level heuristics

We used the same 9 low-level heuristics as given in [7, 11], seven of which (heuristics $h1, h2, h3, h4, h5, h6, h7$) were also used in the tabu search metaheuristic approach of [13], in addition to four variants of chain-moves heuristics. In [13], heuristics were applied in a given order, in conjunction with two tabu lists (one for the nurse that is to be moved and one for the type of shift-patterns utilised). The tabu search algorithm of [13] relied heavily on the chain-moves in order to produce good solutions [1, 2]. Two of the chain-moves heuristics (nurse-chain and shift-chain) were specifically designed to help produce feasible solutions, and two others (nurse-chain and shift-chain) to improve on the penalty cost [13]. The 9 low-level heuristics that we use are:

h1 : Change the shift-pattern of a random nurse.

h2 : Same as [h1] but the acceptance criterion is '1st improving $InFeas$'.

h3 : Same as [h1] but '1st improving $InFeas$ and no worsening of PC'.

h4 : Same as [h1] but '1st improving PC'.

h5 : Same as [h1] but '1st improving PC and no worsening of $InFeas$'.

h6 : Change the shift-pattern type (i.e from day to night or vice versa) of a random nurse, if solution unbalanced.

h7 : Same as [h6] but the aim is to restore balance. That is change a day shift-pattern with a night one if night is unbalanced and vice-versa. If both days and nights are unbalanced a swap of shift-pattern type for a pair of nurses, one working days and the other working night is considered. The nurse working day is assigned a night shift-pattern and the nurse working night is assigned a day shift-pattern.

h8 (Change-and-keep1): This heuristic finds the first move which improves PC by changing the shift-pattern of a nurse and assigning the removed shift-pattern to another nurse.

h9 (Change-and-keep2): Same as [h8], but only considers moves which do not worsen $InFeas$.

The above low-level heuristics are much simpler and easier to code than the chain-moves of [13]. They are all based around changing, or swapping one or two shift-patterns.

3.3 Computational results

Our tabu-search algorithm was coded in Micosoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running under Microsoft Windows 2000 version 5. In both figure 3 and figure 4, we compare results of the tabu-search hyperheuristic (HH) with those obtained using the indirect genetic algorithm of [2] (GA). Both algorithms start with an initial solution obtained by assigning a random feasible shift-pattern to each nurse as in [2, 13]. This rarely leads to a feasible solution. For each of the 52 real-world instances, both algorithms are run 10 times and average results given. The stopping condition for the GA is 30 generations without improvement (population size is 100). The stopping condition for the hyperheuristic was 8000 iterations. In terms of feasibility we report the proportion of feasible solutions obtained at the end of each of the 10 runs (figure 3). Therefore a proportion of 1 means that a feasible solution was produced at the end of each run. In terms of cost we give the average cost over the 10 runs (figure 4).

In the original problem, there are 3 grade-bands for nurse qualifications. We see that in terms of feasibility (top-left chart of figure 3) the hyperheuristic (FeasHH) outperforms the genetic algorithm (FeasGA). The hyperheuristic, which was not designed for this problem, always produces feasible solutions thus making it

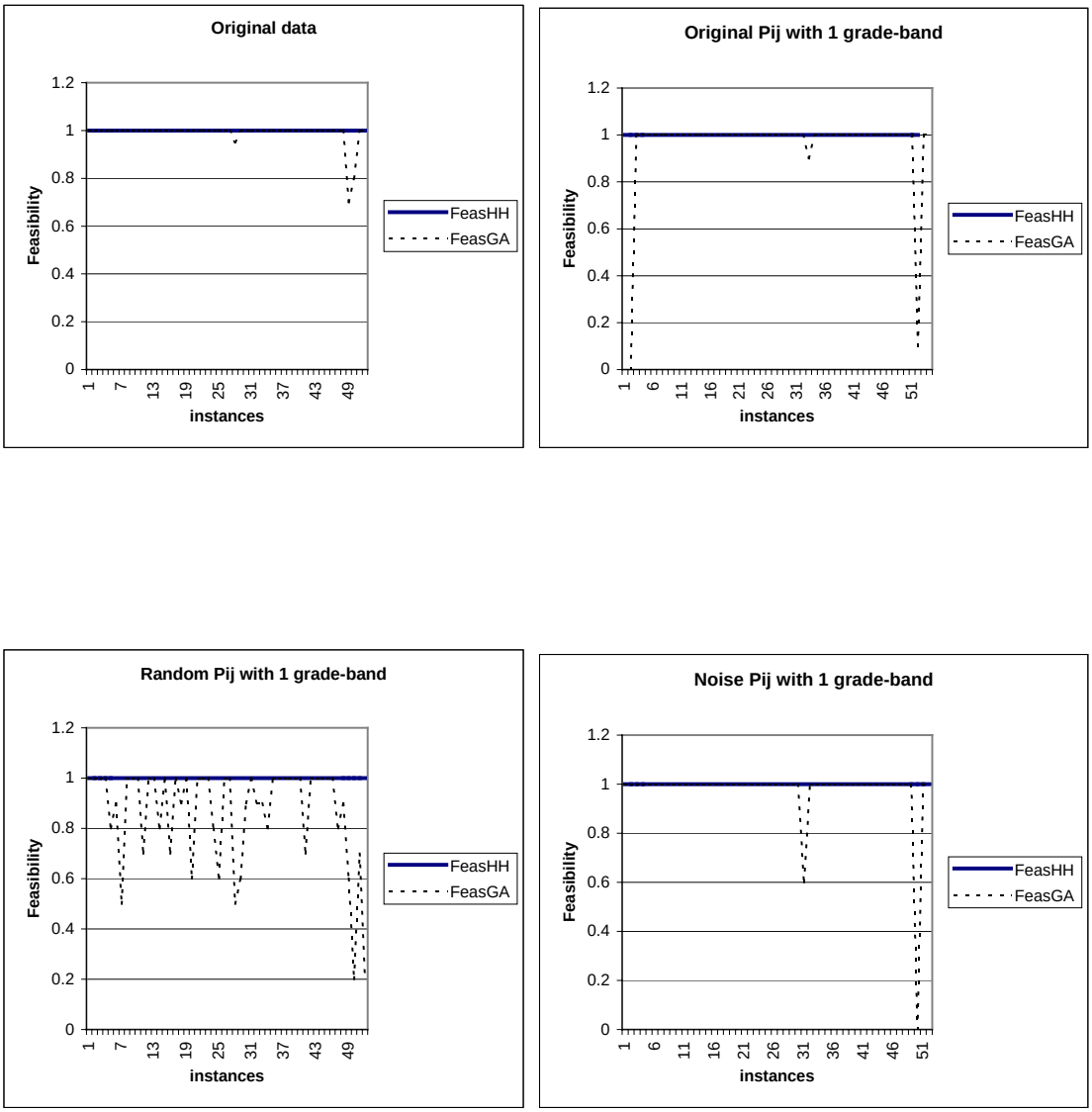
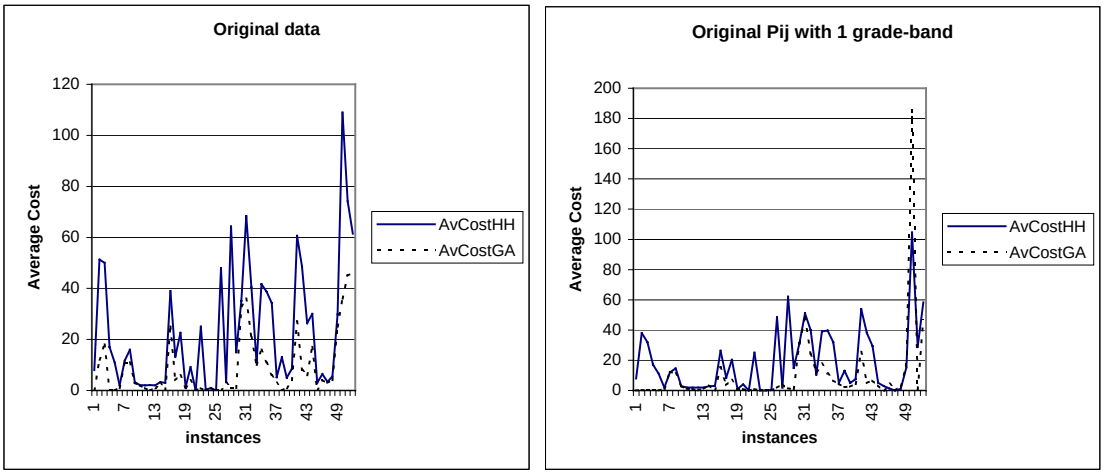


Figure 3: Feasibility study for the nurse scheduling problem. Each chart corresponds to a set of 52 problem instances.



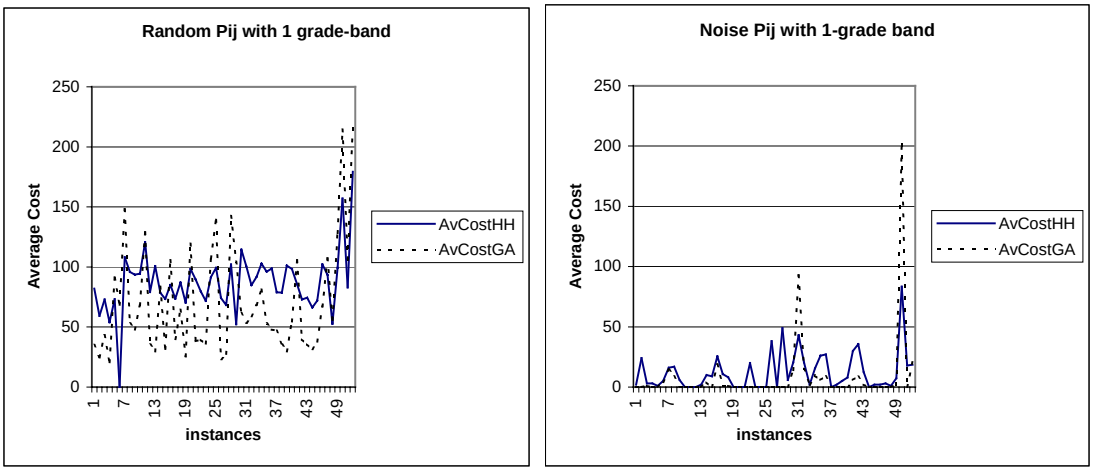


Figure 4: Cost study for the nurse scheduling problem. Each chart corresponds to a set of 52 problem instances.

more reliable than the genetic algorithm which was specifically designed for this problem. In terms of cost however (top-left chart of figure 4) the genetic algorithm outperforms the hyperheuristic, though in many cases the hyperheuristic was also capable of producing solutions of acceptable quality in the same range as those of the GA. Considering that the GA exploited problem-specific information this should not come as a surprise.

To test the generality of both algorithms for this problem, we made the following perturbations to the original data. All nurses are now given the same grade of 1, which means that all nurses now have the same qualification level. 3 sets of problem instances were created: one with the original P_{ij} costs, one where each P_{ij} is randomly generated uniformly in interval $[0, 100]$ and finally one where some random noise uniformly taken in interval $[-10, 10]$ is added to each original P_{ij} (if the resulting Noise P_{ij} is below 0 it is set to 0 and if the resulting Noise is above 100 it is set to 100). In both figures 3 and 4 we give results in terms of feasibility and cost respectively. In each figure, results for Original P_{ij} are in the top-right corner, those for Random P_{ij} are in the bottom-left corner and finally those for Noise P_{ij} are in the bottom-right corner.

In terms of feasibility we see that the hyperheuristic, again, produces feasible solutions for each of the 10 runs and for each of the 3×52 instances of the problem. The performance of the genetic algorithm deteriorates, especially when using Random P_{ij}. In particular, the GA seems to struggle with instance 50 for all 3 sets of perturbed problem instances. Very strikingly the GA could not produce any feasible solution for each of the 10 runs of instance 50 when using Noise P_{ij}. This reveals the brittleness of this tailor-made metaheuristic as opposed to the robustness of the general hyperheuristic method which was not designed with this particular nurse scheduling problem in mind.

In terms of cost, the GA again outperformed the hyperheuristic for all 3 sets of problem instances. We note however that the gap between the GA and the hyperheuristic decreases substantially. For instances in the original data, the GA costs were lower than the hyperheuristic (HH) ones for all 52 instances. However, the GA achieved lower costs than the hyperheuristic for only 37 instances in Original P_{ij} (with 1 grade-band), 29 instances in Random P_{ij}, and 38 instances for Noise P_{ij}. It should be noted, however, that when a feasible solution has not been found in the GA we give a conventional penalty cost value of 255 to the resulting infeasible solution (as used by the author of [2]). Note that the hyperheuristic was able to produce optimal costs for each of the 10 runs of 12 instances when using Noise P_{ij}. This again suggests that the GA is a little brittle while the HH seems to be steadily improving.

Overall it can be argued that the hyperheuristic is capable of producing solutions of acceptable quality for the nurse scheduling problem. Our claim in this paper is that the approach is more general than the current state-of-the-art. We now demonstrate this by employing it on a very different scheduling problem: university course timetabling.

4 An application to university course timetabling

4.1 Problem description

This is one of the application problems considered in the Metaheuristic Network[1], a European Commission project undertaken jointly by 5 European academic institutions, in order to investigate the performance of various metaheuristic techniques on different combinatorial optimisation problems[2]. Here we consider the same timetabling problem as that described in [32, 34]. The problem is a representation of a typical university course timetabling problem. A problem instance generator creates a number of problem instances with different characteristics for different values of given parameters. The problem consists of a set L of events (e.g. lectures, seminars, etc.) to be scheduled in 45 timeslots corresponding to 5 days of 9 hours each, a set R of rooms in which events can take place, a set S of students who attend the events, and a set F of features satisfied by rooms and required by events (e.g. a given event e might require a room equipped with an overhead projector, another might require a room equipped with sound system, video conference etc.).

[1] http://www.metaheuristics.net

[2] The work reported in this paper is not part of the European project.

Each student attends a number of events and each room has a maximum capacity. A feasible solution for the problem is one for which each event has been assigned a timeslot and a room in such a way as to respect the following hard constraints:

1. No student can be scheduled to more than one event at a time;
2. The room meets all features required by the event;
3. Room capacity is respected;
4. No more than one event is allowed per room and per timeslot.

In addition, each solution (whether feasible or not) is penalised equally for each occurence of the following soft constraint violations:

1. A student has a class in the last timeslot of the day;
2. A student has more than 2 classes in a row;
3. A student has only one class on a day.

We use the following evaluation function E for the tabu-search hyperheuristic $E = 1000 \times Hcv + Scv$, where Hcv is the number of hard constraint violations and Scv the number of soft constraint violations. Again we attach a higher weight to Hcv over Scv so as to achieve feasibility, which is, of course, more important than cost. We use the solution representation of [32, 34]. A solution consists of a vector of length $|L|$ where the positions correspond to the events (position j correspond to event e_j for j = 1,...,$|L|$). The value of each position is an integer number between 1 and 45 corresponding to the timeslot index to which the corresponding event has been assigned. For example vector (45, 2, ..., 20) means that event e_1 is assigned to timeslot 45, e_2 to timeslot 2, ..., and $e_{|L|}$ to timeslot 20. As in [32, 34] room assignment is dealt with seperately by way of a matching algorithm which is applied to the solution every time it is altered by any of the low-level heuristics that are outlined below.

4.2 Low-level heuristics

To allow for a fair comparison, we use the following 6 low-level heuristics which were used in [32, 34].

h1 : Move a random event from its current timeslot to a random one.

h2 : Same as [h1] but '1st improving Hcv'.

h3 : Same as [h1] but '1st improving Scv and no worsening of Hcv'.

h4 : Swap the timeslots of two random events.

h5 : Same as [h4] but '1st improving Hcv'.

h6 : Same as [h4] but '1st improving Scv and no worsening of Hcv'.

As for the nurse scheduling problem of the previous section, the low-level heuristic are relatively simple to implement.

Set	HH	RRLS [34]	ANT [34]
S1	1/2.2 /**1**	8	1
S2	1/3 / **2**	11	3
S3	1/1.4 / **0**	8	1
S4	1/1.8 / **1**	7	1
S5	1/0.2 / **0**	5	0
M1	1/179 / **146**	199	195
M2	1/197.6 / **173**	202.5	184
M3	1/295.4 / 267	77.5% Inf	**248**
M4	1/180 / 169	177.5	**164.5**
M5	0.8/388.5 / 303	100% Inf	**219.5**
L	0.2/1166 / 1166	100% Inf	**851.5**

Table 1: Comparison between the hyperheuristic (HH) and the metaheuristics RRLS and ANT. In HH column the format is proportion of feasible solutions in 5 runs / average Scv for feasible solution / best Hcv in all runs. In column RRLS 77.5% Inf and 100% Inf correspond to the proportion of infeasible solutions in 40 runs. Otherwise (for both ANT and RRLS) feasible solutions were found at the end of each run. The best results are shown in bold.

4.3 Computational results

Because of the already existing tabu-search algorithm, we coded the low-level heuristics in Micosoft Visual C++ version 6 under Windows, and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running under Microsoft Windows 2000 version 5. In [34] both a local search method which uses the above 6 low-level heuristics and an ant algorithm were applied to the problem. Both algorithms started with an initial solution generated by randomly assigning a timeslot to each event. This rarely leads to a feasible solution (i.e. a solution with $Hcv = 0$). The hyperheuristic solution also starts from the same solution so as to allow for a fair comparison. Three types of problem instances were created using the problem instance generator. In [34], 11 instances were generated, 5 small (100 events, 5 rooms and 5 features), 5 medium (400 events, 10 rooms and 5 features) and 1 large (400 events, 10 rooms and 10 features). Here we propose to apply the tabu-search hyperheuristic to these 11 instances. Comparison is made in terms of number of evaluations, rather than CPU times.

In Table 1, we present results for our tabu-search hyperheuristic (HH) as well as those for the local search method (RRLS) and the ant algorithm (ANT) which are both described in [34]. The number of evaluations for both RRLS and ANT was approximately 200000. The number of evaluations for HH is 12000 for small, 1200 for medium and 5400 for large, hence the hyperheuristic used substantially fewer evaluations than both RRLS and ANT. In [34], RRLS and ANT were each run 50 times on small instances, 40 times on medium instances, and 10 times on the large instance (using different random seeds). Here we ran HH five times with different seeds. We therefore propose to compare the best HH result in 5 runs with the average results from RRLS and ANT as given in [34].

In terms of feasibility (Hcv), we see that HH was able to produce a feasible solution for each problem, whereas RRLS did not. In particular, RRLS only produced 9 feasible solutions in 40 runs on instance M4. Also RRLS could not produce any feasible solutions in 40 runs on M5. RRLS could not produce any feasible solutions in 40 runs on L (large instance) either. ANT was able to produce feasible solutions for all instances. It would appear that overall the tabu-search hyperheuristic, which was not specifically designed for this problem, performs as well as ANT and outperforms RRLS both of which were specifically designed for this particular problem. It can be seen that the tailor-made ANT algorithm is 'beaten' (or has equivalent results) for all of the small problems and two of the medium problems by the tabu-search hyperheuristic. However, the ANT algorithm produces better results on three of the medium problems and the large problem.

These observations are in line with what we would expect. As the problems get larger, the problem specific features of the tailor-made algorithm begin to give it an 'edge'. However, this edge comes at the expense of having to specially develop an algorithm for a particular problem. The tabu search hyperheuristic is a more general approach which is more widely applicable. It is worth noting that even when it is beaten, the tabu search hyperheuristic is producing results which are reasonably close to the tailor-made algorithm.

In terms of cost (Scv), we see that the best solution produced by HH in 5 runs was better than the average solution of RRLS in most instances. Also for most instances HH is better than or equal to ANT. Again the hyperheuristic proves to be highly competitive with two tailor-made metaheuristics developed for this problem.

It should be noted that for both the nurse scheduling problem and the university course timetabling problem, we used the same parameters of the tabu-search hyperheuristic as given in [7]. We conclude that the hyperheuristic does not seem to be very sensitive to the variety of problem and problem instances considered [7].

5 Conclusions

We have investigated the use of a hyperheuristic approach and applied it to two very different scheduling problems. In our framework, heuristics compete against one another in order to be selected by the hyperheuristic. The rules for competition are inspired by the principles of reinforcement learning [21, 36]. Our hyperheuristic is also equipped with a dynamic tabu list of heuristics which are excluded from the selection at certain times.

The application of the tabu-search hyperheuristic to a highly-constrained real-world problem of scheduling nurses was very successful in terms of producing feasible solutions for the hospital. When compared to a genetic algorithm specifically tailored for the nurse scheduling problem [2] we observed that the tabu-search hyperheuristic outperformed the GA in terms of feasibility. In terms of cost the GA outperformed the hyperheuristic. However, despite the fact that the hyperheuristic was not specifically designed for this problem, solutions produced were of acceptable quality and were certainly competitive with the GA (with optimality even achieved in many cases). Further experiments on three sets of 52 perturbed problem instances revealed the brittleness of the GA, especially in terms of feasibility, while the hyperheuristic tended to be consistently improving, thus closing the gap between the cost performance of both algorithms. Although the tabu-search hyperheuristic produced solutions which were competitive with the GA, the purpose of this research was not to generate a method which could produce 'better' solutions than a specially tailored algorithm on a narrow range of benchmark problems. Instead the goal was to develop an approach which was competitive with the current (problem specific) state of the art but which could be employed on *different* scheduling problems.

In order to demonstrate the wider applicability of this method, the tabu-search hyperheuristic was also applied to a university course timetabling problem. In terms of both feasibility (number of hard constraint violations) and cost (number of soft constraint violations) our hyperheuristic proved to be very competitive with the state-of-the-art techniques for this problem. On seven out of eleven benchmark problems, the tabu-search hyperheuristic, which was not designed with this problem in mind, produced solutions which were better than or equal to those produced using a sophisticated local search method and an ant algorithm, both of which were tailor-made algorithms for this problem [34].

Our tabu-search hyperheuristic is a more general approach. All it requires is a series of simple, easy to implement low-level heuristics. The system (tabu-search hyperheuristic) will decide which of the heuristics to use. Is a heuristic any good? If you believe so, throw it into the mix! The tabu-search hyperheuristic will search the problem space of low-level heuristics with the aim of using the right one under the right conditions.

Our tabu-search hyperheuristic used the same parameter settings for both problems as given in [7]. This confirms the fact that our hyperheuristic does not appear to be particularly sensitive to parameter settings when using a variety of problems and problem instances [7]. It would appear that the proposed

hyperheuristic approach has a good potential for generality across various problems and would require little development time, without substantial deterioration in solution quality as demonstrated in this paper. In fact our hyperheuristic was able to outperform published problem-specific methods on some instances of the university course timetabling problem.

Acknowledgements

We would like to thank Dr Uwe Aickelin and Dr Kath Dowsland for running the GA on all perturbed 3×52 problem instances of the nurse scheduling problem of section 3 and for their valuable comments.

We are also grateful to the UK Engineering and Physical Sciences Research Council (EPSRC) who funded this research (Grant reference number GR/N36837/01).

References

[1] U. Aickelin. *Genetic algorithms for multiple-choice optimisation problems*. PhD thesis, European Business Management School, University of Wales Swansea, September 1999.

[2] U. Aickelin and K. A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3:139–153, 2000.

[3] E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *Selected Papers of the 2nd Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'98)*, Lecture Notes in Artificial Intelligence, pages 186–194. Springer, Berlin Heidelberg New York, 1998.

[4] E. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, November/December 2001.

[5] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of metaheuristics*, chapter 16, Hyper-heuristics: an emerging direction in modern search technology, pages 457–474. Kluwer Academic Publishers, 2003.

[6] E. K. Burke, B. L. MacCarthy, S. Petrovic, and R. Qu. Knowledge discovery in a hyper-heuristic for course timetabling using case based reasoning. In *Selected papers of the 4th International Conference on the Practice And Theory of Automated Timetabling (PATAT 2002)*, Lecture Notes in Computer Science Vol 2740. Springer, 2003. To appear.

[7] E. K. Burke and E. Soubeiga. Scheduling nurses using a tabu-search hyperheuristic. In *Proceedings of the 1st Multi-disciplinary International Scheduling conference: Theory and Applications (MISTA 2003)*, August 2003. To appear.

[8] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Congress on Evolutionary Computation, CEC'02*, pages 1185–1190, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 12-17 2002.

[9] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling PATAT'2000*, Lecture Notes in Computer Science Vol 2079, pages 176–190, 2000.

[10] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Second European Conference on Evolutionary Computing for Combinatorial Optimisation, EvoCop 2002*, Lecture Notes in Computer Science, pages 1–10, Kinsale, Ireland, April, 3-51 2001. Springer.

[11] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In *Parallel Problem Solving from Nature VII, PPSN 2002*, Lecture Notes in Computer Science Vol 2439, pages 851–860, Granada, Spain, September, 7-11 2002. Springer-Verlag.

[12] W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, (117), 1963.

[13] K. A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.

[14] H.L. Fang, P. Ross, and D. Corne. A promising hybrid g.a. / heuristic approach for open-shop scheduling problems. In A. Cohn, editor, *Eleventh European Conference on Artificial Intelligence*, pages 590–594. John Wiley & Sons Ltd, 1994.

[15] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institute of Technology, May 10-12 1961.

[16] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.

[17] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publisher, 2003.

[18] E. Hart and P. Ross. A heuristic combination method for solving job-shop scheduling problems. In A. E. Eiben, T. Back, M. Schoenauer, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature V*, Lecture Notes in Computer Science Vol 1498, pages 845–854. Springer-Verlag, 1998.

[19] J.Mockus, W. Eddy, A. Mockus, L. Mockus, and G. Reklaitis. *Bayesian Heuristic Approach to Discrete and Global Optimization*. Kluwer Academic Publishers, 1997.

[20] J.Mockus and L.Mockus. Bayesian approach to global optimization and applications to multi-objective constrained problems. *Journal of Optimization Theory and Applications*, 70(1):155–170, 1991.

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[22] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hyperheuristics. In *Fourth Asia-Pacific Conference on Simulated Evolution And Learning, SEAL 2002*, pages 667–671, Orchid Country Club, Singapore, November, 17-22 2002. NTU.

[23] J. Mockus. *Bayesian Approach to Global Optimization*. Kluwer Academic Publishers, 1989. 254 pages and a disk with the software system.

[24] A. Nareyek. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In *Fourth Metaheuristic International Conference MIC'2001*, pages 211–215, Porto, Portugal, July 16-20 2001.

[25] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M.Resende and J. Pinho de Sousa, editors, *METAHEURISTICS: Computer Decision Making*. Kluwer, 2003.

[26] I. P. Norenkov. Scheduling and allocation for simulation and synthesis of cad system hardware. In *Proceedings EWITD 94, East-West International Conference, ICSTI*, pages 20–24, Moscow, 1994.

[27] I. H. Osman and G. Laporte. Meta-heuristics: a bibliography. *Annals of Operations Research*, 63:513–628, 1996.

[28] S. Petrovic and R. Qu. Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling. In *Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information & Engineering Systems and Applied Technologies (KES 2002)*, volume 82, pages 336–340, Milan, Italy, September 16-18 2002.

[29] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern Heuristic Search*. John Wiley and Sons, 1996.

[30] C. R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. Blackwell, Oxford, 1993.

[31] P. Ross, S. Schulenburg, J. G. Marin-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In *Proceedings of the Genetic and Evolutionary Computation COnference, GECCO'02*, pages 942–948. Morgan-Kauffman, 2002.

[32] O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, K. Socha, and B. Paechter. A local search for the timetabling problem. In *Proceedings of the 4th International Conference on the Practice And Theory of Automated Timetabling, PATAT 2002*, pages 124–127, August 2002.

[33] J. D. Schaffer and L. J. Eshelman. Combinatorial optimisation by genetic algorithms: The value of the genotype/phenotype distinction. In *First International Conference on Evolutionary Computation and its Applications EvCA'96*, pages 110–120, Moscow, RUSSIA, June 24-27 1996. Presidium of the Russian Academy of Sciences, Springer-Verlag.

[34] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In *Proceedings of the 3rd International Workshop on Ant Algorithms, ANTS 2002*, Lecture Notes in Computer Science Vol 2463, pages 1–13. Springer, September 2002.

[35] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[36] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[37] H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In *Genetic and Evolutionary Computation COnference, GECCO'99*, pages 635–642, 1999.

[38] S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-heuristics: advances and trends in local search paradigms for optimisation*. Kluwer Academic Publishers, 1999.

[39] D. Wolpert and W. G. MacReady. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.