Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling

Peter Cowling[2], Graham Kendall[1], and Eric Soubeiga[1,⋆]

[1] ASAP Research Group, School of Computer Science and IT
University of Nottingham, Nottingham NG8 1BB, UK
`{gxk,exs}@cs.nott.ac.uk`
[2] MOSAIC Research Group, Department of Computing, University of Bradford
Bradford BD7 1DP, UK
`Peter.Cowling@scm.brad.ac.uk`

Abstract. A *hyperheuristic* is a high-level heuristic which adaptively chooses between several low-level knowledge-poor heuristics so that while using only cheap, easy-to-implement low-level heuristics, we may achieve solution quality approaching that of an expensive knowledge-rich approach, in a reasonable amount of CPU time. For certain classes of problems, this generic method has been shown to yield high-quality practical solutions in a much shorter development time than that of other approaches such as tabu search and genetic algorithms, and using relatively little domain-knowledge. Hyperheuristics have previously been successfully applied by the authors to two real-world problems of personnel scheduling. In this paper, a hyperheuristic approach is used to solve 52 instances of an NP-hard nurse scheduling problem occuring at a major UK hospital. Compared with tabu-search and genetic algorithms, which have previously been used to solve the same problem, the hyperheuristic proves to be as robust as the former and more reliable than the latter in terms of solution feasibility. The hyperheuristic also compares favourably with both methods in terms of ease-of-implementation of both the approach and the low-level heuristics used.

Keywords: Hyperheuristic, Heuristic, Personnel Scheduling, Nurse Scheduling.

1 Introduction

Personnel scheduling deals with the allocation of timeslots and possibly locations and other resources to people. This problem has been extensively addressed in the literature over the past 30 years with a survey in almost every decade [3,11,4]. Very often the problem is solved using heuristics. For instance Schaerf [10] tackled a high school timetabling problem formulated as a mathematical programme. He defined two types of neighbourhood moves, the atomic move which swaps two classes of the same lecturer which are scheduled in two different timeslots, and

---

[⋆] Corresponding author

J.J. Merelo Guervós et al. (Eds.): PPSN VII, LNCS 2439, pp. 851–860, 2002.

the double move as a pair of atomic moves. The former is associated with a tabu search and the latter with a randomised nonascendent search (RNA). Both methods are then used alternately. The algorithm produced timetables better than the manual ones for various types of schools. Levine [9] used a hybrid genetic algorithm (GA) to solve an airline crew scheduling problem. The GA was hybridised with a local search heuristic which tries to repair infeasibilities in the solution. Computational experiments compared the hybrid GA with branch-and-cut and branch-and-bound algorithms. Both these latter algorithms produced better solutions than the hybrid GA. Often heuristic methods used to solve personnel scheduling problems make use of sophisticated metaheuristic techniques and problem-specific information to arrive at a good solution. This is the case in [8] and [2] which respectively report the use of tabu search and a genetic algorithm for nurse scheduling.

It is precisely in this context that we proposed a *hyperheuristic* approach [5] as a heuristic that operates at a higher level of abstraction than current metaheuristic approaches. A hyperheuristic controls a set of simple, knowledge-poor, low-level heuristics (for example change, swap, add and drop moves). At each decision point the hyperheuristic must choose which low-level heuristic to apply, without recourse to domain knowledge. Hence we may use hyperheuristics in cases where little domain-knowledge is available (e.g. when dealing with a new, poorly understood or unusual problem) or when a solution must be produced quickly (e.g for prototyping). A hyperheuristic could be regarded as an ‘off-the-peg’ method as opposed to a ‘made-to-measure’ bespoke metaheuristic. It is a generic and fast method, which should produce solutions of at least acceptable quality, based on a set of cheap and easy-to-implement low-level heuristics. In order to apply a hyperheuristic to a given problem, we need only a set of low-level heuristics and one or more measures for evaluating solution quality. In [5,6] and [7] respectively a choice function hyperheuristic of the same type as in section 4 was successfully applied to a real world problem of scheduling business meetings, and to a problem of scheduling undergraduate students’ project presentations at a UK academic institution. In this paper, we use our hyperheuristic technique to solve another real-world problem, that of scheduling nurses at a major UK hospital. The problem has been previously solved using tabu search [8] and genetic algorithms [2]. It is our aim to demonstrate that hyperheuristics are not only readily applicable to a wide range of scheduling and other combinatorial optimisation problems, but also can provide very good-quality solutions comparable to those of knowledge-rich sophisticated metaheuristics, while using less development time and simple, easy-to-implement low-level heuristics. In sections 2, 3, 4 and 5 we present respectively the nurse scheduling problem, the solution methods used, experimental results, and conclusions.

2 The Nurse Scheduling Problem

The problem is to create weekly schedules for wards containing up to 30 nurses at a major UK hospital. These schedules must respect working contracts and meet the demands (i.e number of nurses of different grades required) for each day-shift and night-shift of the week, whilst being perceived as fair by the nurses themselves. Nurses work either day-shifts, divided into 'earlies' and 'lates', or night-shifts in a given week. A full week's work typically includes more days than nights[1]. As mentioned in [8], the problem can be decomposed into 3 independent stages. Stage 1 uses a knapsack model to check if there are enough nurses to meet demands. Additional nurses are needed for Stage 2 otherwise. This latter stage is the most difficult and is concerned with the actual allocation of the weekly shift-pattern schedules to each nurse. Then stage 3 uses a network flow model to assign those on day-shifts to 'earlies' and 'lates'. As in [8] and [2] we limit ourselves to the highly constrained problem in stage 2, as stages 1 and 3 can be solved quickly using standard knapsack and network flow algorithms. The stage 2 problem is described as follows. Each possible weekly shift-pattern for a given nurse can be represented as a 0-1 vector of 14 elements (7 day-shifts and 7 night-shifts). A '1'/'0' in the vector represents a day or night 'on'/'off'. For each nurse there is a limited number of shift-patterns corresponding to the number of combinations of the number of days s/he is contracted to work in a week[2]. There are typically between 20 and 30 nurses per ward, 3 grade-bands, and 411 different (F/T and P/T) shift-patterns. Based upon the nurses' preferences, the recent history of patterns worked, and the overall attractiveness of the pattern, a penalty cost is associated to each assignment nurse-shift pattern, values of which were set after agreement with the hospital, ranging from 0 (ideal) to 100 (undesirable) -See [8] for further details. Our decision variables are denoted by x_{ij} assuming 1 if nurse i works shift-pattern j and 0 otherwise. Let parameters g, n, s be the number of grades, nurses and possible shift-patterns respectively. a_{jk} is 1 if shift-pattern j covers shift k, 0 otherwise. b_{ir} is 1 if nurse i is of grade r or higher, 0 otherwise. p_{ij} = penalty cost of nurse i working shift-pattern j; S_{kr} = demand of nurses of grade r or above on day/night (i.e shift) k; and $F(i)$ = set of feasible shift-patterns for nurse i. We may then formulate the problem as follows:

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

s.t.

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

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

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

---

[1] e.g. a full-time nurse works 5 days or 4 nights, whereas a part-time nurse works 4 days or 3 nights, 3 days or 3 nights, and 3 days or 2 nights.

[2] For example a F/T nurse contracted to work either 5 days or 4 nights has a total of $C_7^5 = 21$ feasible day shift-patterns and $C_7^4 = 35$ feasible night shift-patterns.

Equations (1), (2) and (3) express respectively the objective of minimising the overall penalty cost associated with the nurses' desirability for the shift-patterns, the constraint that each nurse should work exactly one shift-pattern, and the demand constraints. It should be noted that b_{ir} is defined in such a way that higher-grade nurses can substitute for those at lower grades if needed. The problem is NP-hard [2] and instances typically involve between 1000 and 2000 variables and up to 70 constraints. As noted in [2], the difficulty of a given instance depends upon the shape of the solution space, which in turn depends on the distribution of the penalty cost (p_{ij}) and their relationship with the set of feasible solutions. In this paper we consider 52 data instances, based on three wards and corresponding to each week of the year. These 52 instances, as a whole, feature a wide variety of solution landscapes ranging from easy problems with many low-cost global optima scattered througout the solution space, to very hard ones with few global optima and in some cases with relatively sparse feasible solutions [2]. Optimal solutions are known for each instance as the problem was solved using a standard IP package. However some instances remained unsolved after 15 hours of (Pentium II 200 Mhz PC) run-time. Further experiments with a number of descent methods using different neighbourhoods, and a standard simulated annealing were conducted unsuccessfully, failing to obtain feasibility [2]. The most successful approach which works within the low CPU time available so far is a tabu search which uses chain-moves whose design and implementation were highly problem and instance specific as these moves relied on the way the different factors affecting the quality of a schedule were combined in the p_{ij} as noted in [2]. In [2] a GA which did not make use of chain-moves was also used to solve the problem. Failure to obtain good solutions led to the use of a co-evolutionary strategy which decomposed the main population into several co-operative sub-populations. Problem structure was incorporated in both the way the sub-populations were built, and the way partial solutions were recombined to form complete ones. As a result, the applicability of the co-evolutionary strategy is, likewise, limited to problems with a similar structure.

Here we propose to solve the nurse scheduling problem using a high-level hyperheuristic approach which has been successfully applied to two rather different real-world problems of personnel scheduling. The evaluation function[3] used by the hyperheuristic distinguishes between 'balanced' and 'unbalanced' solutions [8,2]. Effectively, since nurses work either days or nights it appears that in order for a given solution to be feasible, (i.e enough nurses covering all 14 shifts at each grade) the solution must have sufficient nurses in both days and nights separately. Formally, a solution is balanced in days (or nights) at a given grade 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. In fact, a solution cannot be made feasible until it is balanced [8,2]. We define $Infeas = \sum_{r=1}^{g}(\rho \times Bal_r + 1)\sum_{k=1}^{14} max\left(\left[S_{kr} - \sum_{i=1}^{n}\sum_{j=1}^{s} b_{ir}a_{jk}x_{ij}\right], 0\right)$,

---

[3] known as fitness function in the GA literature

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 parameter set to 5, 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. Based on this, we define the evaluation function $E = PC + C_{demand} InFeas$ with C_{demand} a weight associated to $InFeas$ 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][4]. It is interesting to note that while in [8] unbalanced solutions are repaired, in [2] they are instead avoided through the use of incentives/disincentives to reward/penalise balanced/unbalanced individuals in the population. Here we opt for the former approach and use the same 'balance-restoring' low-level heuristic used in tabu search of [8]. As described in section 4, this low-level heuristic uses a 'change' and a 'swap' type of move. We next describe our hyperheuristic method.

3 A Choice-Function Hyperheuristic Technique

Our hyperheuristic is based upon a *Choice-Function* which adaptively ranks the low-level heuristics. Originally [5], the choice function is determined based on information with regards to individual performance of each low-level heuristic (f_1), joint performance of pairs of heuristics (f_2), and the amount of time elapsed since the low-level heuristic was last called (f_3). Thus we have $f_1(N_j) = \sum_n \alpha^{n-1} (\frac{I_n(N_j)}{T_n(N_j)})$ and $f_2(N_j, N_k) = \sum_n \beta^{n-1} (\frac{I_n(N_j, N_k)}{T_n(N_j, N_k)})$ where $I_n(N_j)/$ $I_n(N_j, N_k)$ (respectively $T_n(N_j)/T_n(N_j, N_k)$) is the change in the objective function (respectively the number of CPU seconds) the n^{th} last time heuristic N_j was called/called immediately after heuristic N_k. Both α and β are parameters between 0 and 1, reflecting the greater importance attached to recent performance. f_1 and f_2 aim at intensifying the search. The idea behind the expressions of f_1 and f_2 is analogous to the exponential smoothing forecast of their performance [12]. f_3 provides an element of diversification, by favouring those low-level heuristics that have not been called recently. Then we have $f_3(N_j) = \tau(N_j)$ where $\tau(N_j)$ is the number of CPU seconds which have elapsed since low-level heuristic N_j was last called. If the low-level heuristic just called is N_j then for any low-level heuristic N_k, the choice function f of N_k is defined as $f(N_k) = \alpha f_1(N_k) + \beta f_2(N_j, N_k) + \delta f_3(N_k)$. In this expression, the choice function attempts to predict the overall performance of each low-level heuristic. In [6], we presented a different choice function which separately predicts the performance of each low-

---

[4] See [2] for an interesting discussion on the choice of evaluation functions.

level heuristic with respect to each criterion of the evaluation function instead (PC and $Infeas$ in the model above). The choice function f is then decomposed into $f(N_k) = \sum_{l \in \mathbf{L}} f_l(N_k) = \sum_{l \in \mathbf{L}} \left[\alpha_l f_{1l}(N_k) + \beta_l f_{2l}(N_j, N_k) + \frac{\delta}{|L|} f_3(N_k)\right]$ where $\mathbf{L} = \{PC, Infeas\}$ is the set of the evaluation function criteria, and $f_{1l}(N_k)$ (respectively $f_{2l}(N_j, N_k)$) is obtained by replacing $I_n(N_k)$ (respectively $I_n(N_j, N_k)$) with $I_{ln}(N_k)$ (respectively $I_{ln}(N_j, N_k)$) in the expression of $f_1(N_j)$ (respectively $f_2(N_j, N_k)$) above. $I_{ln}(N_k)$ (respectively $I_{ln}(N_j, N_k)$) is the first (respectively second) order improvement with respect to criterion $l \in \mathbf{L}$. Parameter values for α, β and δ are changed adaptively using the procedure in [6]. We will give results for the second approach which works as follows

```
Do
    Choose a search criterion $l$
    - Select the low-level heuristic that maximises $f_l$ and apply it.
    - Update choice function $f_l$'s parameters using the adaptive procedure
Until Stopping condition is met.
```

The probability of choice of criteria PC and $InFeas$ is given by $p_1 = \frac{1}{1+C_{demand}}$ and $p_2 = \frac{C_{demand}}{1+C_{demand}}$ respectively as defined in [6]. We would like to emphasize the fact that the implementation of the hyperheuristic technique was quite fast. In effect the hyperheuristic presented here is a ‘standard’ approach which was successfully applied to two different real-world problems [5,6,7]. The hyperheuristic approach only requires a set of low-level heuristics to be added to the hyperheuristic black box, and a formal means of evaluating solution quality. The way the hyperheuristic works is independent of both the nature of the low-level heuristics and the problem at hand. Hence important savings in development time are made possible by the use of the hyperheuristic framework. Development of the framework itself took over eighteen months. For example in [7] high-quality solutions were initially developed in just over two weeks after meeting with the problem owner. Solution development time for the current problem was one-and-a-half months, due to the larger number of instances to be handled and the development of low-level heuristics for this challenging highly-constrained real-world problem. We show in the next section that, despite such a relatively short development time, the hyperheuristic - even when dealing with a very difficult problem - is capable of finding solutions of good quality comparable to those of bespoke metaheuristics within a reasonable amount of time.

4 Experiments

Both our hyperheuristic and its low-level heuristics were 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 order to compare our results with those of tabu search (TS) and genetic algorithms (GA), our hyperheuristic starts with a solution generated randomly by assigning a random feasible shift-pattern to each nurse as in [8]. All results were averaged over 20 runs. The TS algorithm of [8] used the following 11 low-level heuristics:

[h1] Change the shift-pattern of a random nurse; [h2] Same as [h1] but 1st improving $InFeas$; [h3] Same as [h1] but 1st improving $InFeas$, no worsening of PC; [h4] Same as [h1] but 1st improving PC; [h5] Same as [h1] but 1st improving PC, no worsening of $InFeas$; [h6] Change the shift-pattern type (i.e day/night) of a random nurse, if solution unbalanced; [h7] Same as [h6] but aim is to restore balance[5]; [h8] (shift-chain1): This heuristic considers chains of moves aiming at decreasing both the nurse-shortage in one (under-covered) shift and the nurse-surplus in one (over-covered shift), and leaving the remaining shift unchanged; [h9] (nurse-chain1): Considers chains of moves which move the first nurse in the chain to cover an under-covered shift and move the subsequent nurses to the shift-pattern just vacated by their predecessor in the chain.[6]; [h10] (shift-chain2): Considers a shift-chain of moves aiming at decreasing the penalty cost when the solution is already feasible; [h11] (nurse-chain2): Considers nurse-chains of moves aiming at decreasing the penalty cost when the solution is already feasible[7].

Instead, our hyperheuristic uses 9 low-level heuristics including the first 7 low-level heuristics above and the following: [H8] (Change-and-keep1): This heuristic changes the shift-pattern of a nurse and assigns the removed shift-pattern to another nurse (1st improving PC); [H9] (Change-and-keep2): Same as [H8], but 1st improving PC and no worsening of $InFeas$.

The chain-moves are highly effective moves which were responsible for both feasibility (using shift-chain1 and nurse-chain1) and optimality (using shift-chain2 and nurse-chain2) of the solution in most cases (see [8] for further details). TS can only yield good solutions when equipped with such moves [1,2]. However, as noted in [1,2] these moves are highly problem-dependent and, in fact, instance-type dependent. Unlike in TS, the low-level heuristics used by the hyperheuristic are fewer and much simpler than the chain-moves. They are all based around changing, or swapping one or two shift-patterns, thus reflecting what users usually do in practice [5]. In Table 1, we present the results of our hyperheuristic, along with those of both the direct and indirect GA [1,2] as well as TS [8] and the IP optimal solution [1] for each of the 52 weeks (problem instances) of the year. The stopping condition of the hyperheuristic is 6000 iterations, which corresponds to a CPU time between 44 and 60 seconds on a Pentium II 1000Mhz[8]. We see that for all instances the hyperheuristic is able to find feasible solutions in each of 20 runs. It appears that the hyperheuristic is more reliable than both the direct and the indirect GA in terms of producing practical solutions for the hospital. To confirm the reliability of the hyperheuristic, we ran it on instance 50 (which is a difficult instance for both GA's and appeared to be the most difficult for the hyperheuristic) 100 times and feasibility was again achieved always

---

[5] i.e from day to night 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.

[6] Both [h8] and [h9] chain-moves are defined as paths in a graph. The move is only attempted if the solution is already balanced but not yet feasible.

[7] This time both [h10] and [h11] chains are represented as cycles in a graph.

[8] The TS stopping condition was 1000 moves without overall improvement.

Table 1. Hyperheuristic and metaheuristic performances on the nurse scheduling problem. Results are averaged over 20 runs. Format is proportion of feasible solutions in 20 runs/average penalty cost.

Instances	Hyperheuristic	Direct GA	Indirect GA	Tabu seach	IP cost
Week 1	1/8	1/0	1/0	0	0
Week 2	1/52.8	1/12	1/12	11	11
Week 3	1/50	1/18	1/18	18	18
Week 4	1/17	1/0	1/0	0	0
Week 5	1/11	1/0	1/0	0	0
Week 6	1/ 2	1/1	1/1	1	1
Week 7	1/13.55	0.5/13	1/11	11	11
Week 8	1/14.95	1/11	1/11	11	11
Week 9	1/3.6	0.95/3	1/3	3	3
Week 10	1/5.05	1/1	1/2	1	1
Week 11	1/2	1/1	1/1	1	1
Week 12	1/2	1/0	1/0	0	0
Week 13	1/2	1/1	1/1	1	1
Week 14	1/3.15	1/3	1/3	3	3
Week 15	1/3.05	1/0	1/0	0	0
Week 16	1/40.1	0.95/25	1/25	24	24
Week 17	1/17.6	1/4	1/4	4	4
Week 18	1/20.85	1/7	1/6	7	6
Week 19	1/1.6	1/1	1/1	1	1
Week 20	1/15.45	0.95/5	1/4	4	4
Week 21	1/0	1/0	1/0	0	0
Week 22	1/25.5	1/1	1/1	1	1
Week 23	1/0	0.95/0	1/0	0	0
Week 24	1/1	0.75/1	1/1	1	1
Week 25	1/0.4	1/0	1/0	0	0
Week 26	1/48	0.1/0	1/0	0	0
Week 27	1/3.65	1/2	1/3	2	2
Week 28	1/65.8	1/1	0.95/1	1	1
Week 29	1/15	0.35/3	1/1	2	1
Week 30	1/39.4	1/33	1/33	33	33
Week 31	1/66.9	0.8/66	1/36	33	33
Week 32	1/41.6	1/21	1/21	20	20
Week 33	1/10.6	1/12	1/10	10	10
Week 34	1/42.9	1/17	1/16	15	15
Week 35	1/38.8	1/9	1/11	9	9
Week 36	1/34.85	1/7	1/6	6	6
Week 37	1/8.05	1/3	1/3	3	3
Week 38	1/13.3	1/3	1/0	0	0
Week 39	1/5.1	1/1	1/1	1	1
Week 40	1/9.35	1/5	1/ 4	4	4
Week 41	1/61.3	0.95/27	1/27	27	27
Week 42	1/47.55	1/5	1/8	5	5
Week 43	1/27.35	0.9/8	1/6	6	6
Week 44	1/31.75	0.9/45	1/17	16	16
Week 45	1/5.35	1/0	1/0	0	0
Week 46	1/9.4	0.7/6	1/4	3	3
Week 47	1/3.3	1/3	1/3	3	3
Week 48	1/6.05	1/4	1/4	4	4
Week 49	1/30.4	1/26	0.7/25	24	24
Week 50	1/109.25	0.35/38	0.8/36	35	35
Week 51	1/74.3	0.45/46	1/45	45	45
Week 52	1/62.2	0.75/63	1/46	46	46
Average	1/23.5	0.91/10.8	0.99/9.0	1/8.8	8.7
Run time	< 60 sec	15 sec	10 sec	30 sec	up to hours

within 6000 iterations (less than a minute of CPU time). From this point of view, the hyperheuristic appears to be as robust as TS which, too, always found feasible solutions. The hyperheuristic however has the highest average cost of 23.5, though more than 50% of the instances (27 instances) were solved to within 10% of the optimal solution, including 3 instances (weeks 21, 23 and 24) where optimality is reached on each of 20 runs. Also in 9 instances (Weeks 7, 9, 14, 19, 25, 27, 33, 47 and 48) the optimal solution is hit up to 19 times out of 20 runs, corresponding to a probability of optimality of 0.95. This shows that optimal solutions are indeed, within the reach of the hyperheuristic in spite of its simplicity and that of its low-level heuristics, when compared with the problem and instance-specific information used by the TS (chain-moves) and GA (population decomposition and recombination using problem structure) implementations. In terms of cost, we noted that the hyperheuristic performed well for instances with slack demand-constraints and poorly for those with tight constraints (e.g Weeks 26, 28, 42, 50).

Observations of the frequency of call of the low-level heuristics showed that [h2] is called most often (e.g 37% on average for Week 49), followed by [h6] (e.g 10% on Week 49) and all other heuristics are called between 5% and 9%. It appears that each low-level heuristic has a part to play in the search. Observations of $Infeas$ and PC showed that immediately upon finding a feasible solution (i.e $InFeas = 0$) there was a sudden increase in PC. Similar observations mere made in [8]. Regarding choice-function parameters, the hyperheuristic search used a very high δ and a low α and β, thus confirming the need to diversify the search quite frequently, due to the sparse spread of good solutions in the landscape [2]. This was in total agreement with the graph of the variation of $InFeas$ overtime which featured sudden low peaks of $Infeas = 0$, similar to the 'comb' shape graph of the same function in [8]. Typically values of $InFeas = 0$ never lasted more than 41 heuristic calls (compared to a total of 10000 heuristic calls overall) after they were obtained. Values for α_{InFeas} and β_{InFeas} were relatively higher than those of α_{PC} and β_{PC} clearly showing the greater importance attached to feasibility over lowering PC.

5 Conclusions

We have applied a hyperheuristic to an NP-hard highly-constrained problem of scheduling nurses at a major UK hospital. The problem had previously been solved using tabu search and two genetic algorithms. In terms of solution feasibility, our hyperheuristic proved more reliable than both the direct and indirect genetic algorithms and proved to be as robust as tabu search. In terms of cost, over half of the instances were solved within 10% of optimality. In a few instances the hyperheuristic obtained optimal solutions with probability of up to 1, thus proving that optimality is indeed within the reach of the hyperheuristic, in spite of its simplicity and that of its low-level heuristics when compared to the highly problem-specific information used by both TS and the GA's. Because of their problem-specific considerations, both TS and GA implementations for this problem are limited in their applicability to other problems as opposed to the hyperheuristic which has been successfully applied to two other personnel-scheduling problems [5,6,7]. Moreover, the hyperheuristic does not need any parameter tuning. Hyperheuristics are easy-to-implement and require less domain knowledge than most other heuristic approaches, yet still are able to arrive at good-quality solutions even for very difficult problems within a reasonable amount of CPU and implementation time. It appears that hyperheuristics can be robust and reliable for solving real-world problems of scheduling and optimisation. Ongoing research will investigate other types of hyperheuristics applied to a wider range of real-world problems.

Acknowledgements

We express our gratitude to both Dr Kath Dowsland and Dr Uwe Aickelin for providing us with data and for their valuable support.

References

1. U. Aickelin. Genetic algorithms for multiple-choice optimisation problems. *PhD Thesis, the University of Wales Swansea*, 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. K. Baker. Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, 27(1):155–167, 1976.
4. D. J. Bradley and J. B. Martin. Continuous personnel scheduling algorithms: a literature review. *Journal Of The Society For Health Systems*, 2(2):8–23, 1990.
5. 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*, Springer Lecture Notes in Computer Science, 176-190, 2001.
6. P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. Proceedings of the 4th Metaheuristic International Conference, MIC 2001, 127-131.
7. P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. Proceedings of the 2nd European Conference on EVOlutionary COmPutation, EvoCop 2002. To appear.
8. K. A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.
9. D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers and operations research*, 23(6):547–558, 1996.
10. A. Schaerf. Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man and Cybernetics Part A:systems and Human*, 29(4):368–377, 1999.
11. J. M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, July 1982.
12. S. C. Wheelwright and S. Makridakis. *Forecasting methods for management.* John Wiley & Sons Inc, 1973.