A Time Pre-defined Variable Depth Search for Nurse Rostering

Edmund Burke[1], Timothy Curtois[1], Rong Qu[1], Greet Vanden Berghe[2]

[1]School of Computer Science & I.T., University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham. NG8 1BB. UK
[2]Information Technology Engineering Department, KaHo St. Lieven, Gebr. Desmetstraat 1, 9000 Gent. Belgium

Abstract

A variety of neighbourhood operators have been used in local search and metaheuristic approaches to solving nurse rostering problems. We test and analyse the efficiency of these neighbourhoods on benchmark problems taken from real world scenarios. A variable depth search is then developed based on the results of this investigation. The algorithm heuristically chains together moves and swaps which define the more effective search neighbourhoods. A number of heuristics for creating these chains were developed and we present experiments conducted to identify the best ones. As end users vary in how long they are willing to wait for solutions, a particular goal of this research was to create an algorithm that accepts a user specified computational time limit and uses it effectively. When compared against previously published approaches the results show that the algorithm is very competitive.

1 Introduction

High quality nurse rosters benefit nurses, patients and managers. Patients receive better healthcare if nurses are able to spend more time with them and mistakes are less likely if nurses are not stressed, tired and overworked due to poor scheduling and understaffing. Improved rosters can not only decrease nurse fatigue but also help them maximise the use of their leisure time and satisfy more of their personal requests. From a management point of view, better and more flexible scheduling can help retain nurses and aid recruitment, reduce tardiness and absenteeism, increase morale and productivity and provide better patient service and safety. Costs can also be reduced through having to hire less agency nurses to fill gaps in rosters and lower staff turnover.

Creating rosters, however, is a difficult and challenging search problem which requires the satisfaction of many constraints and the balancing of a variety of requirements. This time consuming and frustrating duty often falls to a head nurse who would rather be concentrating on their primary duty of caring for patients. Regular rescheduling may also be required to deal with staff sickness and absences. Computerised, automated rostering can remove the vast majority of this workload and provides a number of additional benefits including being able to create much higher quality schedules which are fair, impartial and satisfy more preferences. Legal requirements can also be checked, management statistics collected and monthly reports generated, all reducing paperwork. Other bonuses can range from linking to payroll systems to emailing out rosters or publishing them on the web once they are created.

Over the past 30-40 years, many different approaches have been used to solve nurse rostering problems of varying forms and complexity. Methods used include

mathematical programming [6, 28, 35, 39, 44, 47], constraint programming [20, 36, 38], goal programming and multi objective approaches [3, 4, 10, 27] and case-based reasoning [7, 8]. A great variety of local search and metaheuristic approaches have also been applied to the problem, [2, 9, 12, 21, 37, 45] represent just a few recent examples. Many more can be found in the literature reviews of Burke et al. [17] and Ernst et al. [23]. There are, however, very few large-scale neighbourhood searches applied to nurse rostering problems.

As mentioned in a recent survey paper by Ahuja et al. [1] many successful very-large scale neighbourhood search techniques have appeared in various forms in the field of operations research. They commented, for example, that the well known Lin-Kernighan algorithm for the travelling salesman problem can be viewed as a very large-scale neighbourhood search technique. Ahuja et al. categorised very large-scale neighbourhood methods into three similar classes, one of which are variable depth methods. Variable depth searches (including some ejection chain methods [24]) have been effectively applied to a number of optimisation problems, for example the vehicle routing problem [42] and the generalised assignment problem [48]. Many more examples of successful very-large scale neighbourhood searches can be found in the survey paper of Ahuja et al.

One paper that does introduce the application of such techniques to nurse rostering is that of Dowsland [22]. In her approach to providing an automated nurse rostering system, a tabu search is used that oscillates between decreasing cover violation and increasing roster quality. In each of these phases, two types of ejection chains are used. The first consists of a sequence of on/off day swaps between nurses and the other is made up of sequences of swapping week long work patterns between nurses. The chains are able to escape from bad optima that single on/off day or pattern swaps would not be able to escape from.

Another example of using such techniques in solving a nurse rostering problem is the method of Louw et al. [34] who also use an ejection chain approach. The compound move used is similar to Dowsland's chain of on/off day swaps. They noted "*the compound move was able to achieve far superior reductions in the objective function value when compared to any of the elementary move types*".

Very large-scale neighbourhood searches face the problem of exploring an exponentially large neighbourhood. Therefore, the key to developing effective ones is identifying heuristics and other mechanisms which can efficiently narrow or direct the search. This paper presents a variable depth search for nurse rostering and describes the heuristics and other features that make it successful.

In developing this algorithm, we have also created some test instances based on real world problems which are now publicly available and which are intended to become benchmark problems in this area. As has been mentioned in bibliographic surveys of nurse rostering problems, there is a great lack of test problems and benchmarks [17] and, as such, there is little comparative analysis of nurse rostering algorithms in the literature. Such analysis is very important to underpin scientific progress in this area. Although there are understandable reasons for this lack of publicly available problems (for example, there is no such thing as a typical nurse rostering problem and there is often an issue over the confidentiality of data) we hope our presentation of these problems goes some way to starting to fill this void.

In Section 2, the problem is introduced. Section 3 investigates various search neighbourhoods that have been used to solve nurse rostering problems. Section 4 presents the variable depth search and Section 5 contains the results from a number of

experiments using this algorithm. Finally, Sections 6 and 7 discuss conclusions and future work respectively.

2 Problem Description

The data for this problem is based on and is similar to the data used in [12, 15, 16]. The algorithms previously developed for this problem were developed as part of a commercial system and as the data is from a real world environment it has been anonymized and any confidential information removed.

The problem requires the production of non-cyclical schedules which satisfy all hard constraints and as many working preferences and requests as possible. There are so many conflicting constraints and requests that if they were all hard constraints, a feasible solution would generally not exist. Instead, they are modelled as soft constraints and given relative priorities using weights.

In order to provide these data sets as benchmarks, we have made software implementations of these evaluation functions publicly available (http://www.cs.nott.ac.uk/~tec/NRP/). They can be used as a standard and ensure the accuracy of any new results. As well as making these evaluation functions available, we have also provided the application programming interface developed in order to create and test the algorithms we are presenting here. For example, it is possible to download the software for parsing and using the data, displaying, analysing and manually altering solutions and the algorithms themselves.

2.1 Hard Constraints

In a final roster or solution, if all hard constraints are not satisfied then the roster is not feasible and is thus not acceptable. There are three hard constraints:

1. A nurse cannot be assigned more than one of the same shift type per day.
2. Shifts which require certain skills can only be covered by (i.e. assigned to) nurses who have those skills.
3. The shift coverage requirements must be satisfied. For example, if a certain day requires three night shifts then there must be three employees present at that time to work during that shift. Over coverage is not permitted.

2.2 Soft Constraints

All other rules are modelled as soft constraints. Ideally these constraints should be satisfied but it will almost always be necessary to violate some of them in order to provide a feasible solution. When a soft constraint is broken, a penalty is incurred which is proportional to the importance of the constraint and the severity of its violation. The importance of each constraint is set using weights which are user-modifiable integer values. The higher the weight, the more important the constraint is relative to the other constraints.
With permission each employee can specify which constraints they wish to see imposed on their work schedule and also the specific parameters associated with that constraint. For example, one nurse may request not to work more than five consecutive days whereas another may wish to work no more than four consecutive days. Alternatively, some nurses may have part time contracts which stipulate less work than full time employees and so on. The large number and variety of rules which

can be imposed makes it a very flexible system which can be used in many different environments. It also makes it possible to create problem instances which vary significantly in size and complexity. Therefore a problem solver which is robust and effective over a range of instances is paramount.

Some of the constraints which are considered include:

- Maximum number of shifts worked during the scheduling period.
- Maximum and minimum number of hours worked during the scheduling period or per week.
- Maximum and minimum number of consecutive working days.
- Maximum and minimum number of consecutive non-working days.
- Maximum number of a specific shift type worked. For example, maximum zero night shifts for the planning period or a maximum of seven early shifts. This constraint can also be specified for each week. For example, a nurse may request no late shifts for a certain week.
- Maximum number of weekends worked in four weeks (a weekend definition is also a user definable parameter i.e. Friday and/or Monday may be considered as part of the weekend).
- Maximum number of consecutive weekends worked.
- No night shifts before a weekend off.
- No split weekends, i.e. shifts on all days of the weekend or no shifts over the weekend.
- Identical shift types over a weekend. For example, if a nurse has a day shift on Saturday then he/she may prefer to have a day shift on Sunday also.
- Minimum number of days off after night shifts.
- Valid numbers of consecutive shift types. For example, three or four consecutive early shifts may be valid but two or five consecutive early shifts may not.
- Shift type successions. For example, if shift rotation is allowed, is shift type A allowed to follow B the next day?
- Maximum total number of assignments for all Mondays, Tuesdays, Wednesdays… For example, a nurse may request not to work on Wednesdays or may require to work a maximum of two Tuesdays during the scheduling period.
- Avoid a secondary skill being used by a nurse. Sometimes a nurse may be able to cover a shift which requires a specific skill but they may be reluctant to do so as it is not their preferred duty. An example would be a head nurse not wanting to stand in for a regular nurse.

The precise definition and implementation of these constraints can be found in [46].

The evaluation function used to determine the quality of a roster is the sum of all penalties calculated for each soft constraint violation in each nurse's individual schedule. The problem objective is to find a roster with as low a penalty as possible. Of course, due to the often conflicting and large number of constraints there is rarely a perfect roster with penalty zero. Instead, we can think of this as an optimisation problem with the goal of minimizing the roster's overall penalty.

Let:

$penalty_r$ = the penalty for roster r.

$penalty_{r,n}$ = the penalty for the schedule of nurse n in roster r.

N_r = the number of nurses in roster r.

$$penalty_r = \sum_{n=1}^{N_r} penalty_{r,n}$$

The objective is to minimize $penalty_r$ for a given scheduling period with a fixed set of employees, cover requirements and a specific set of soft constraints and weights.

The algorithms presented in this paper are tested on ten instances which vary in the number of nurses, cover requirements, shift types, constraint types and priorities, personal requests and planning horizon. Table 1 provides some more information on the instances.

Instance	Nurses	Shift types	Skill levels	Planning horizon
BCV-1.8.1	8	4	2	28 days
BCV-2.46.1	46	4	1	28 days
BCV-3.46.1	46	3	1	26 days
BCV-4.13.1	13	4	2	29 days
BCV-5.4.1	4	4	1	28 days
BCV-6.13.1	13	4	2	30 days
BCV-7.10.1	10	6	1	28 days
BCV-8.13.1	13	4	2	28 days
BCV-A.12.1	12	4	2	31 days
ORTEC01	16	4	1	31 days

Table 1 Problem Instances

Data sets one to eight are all based on real world data. Data set BCV-A.12.1 is a fictional test problem that uses all the possible constraint types available and contains many conflicting requests. ORTEC01 is instance 12 for the problem presented in [14] (the instance which the extra tests were performed on and the one commonly used by ORTEC). To model the problem with this system, some of its hard constraints have been changed to soft constraints with weights of 10,000. Therefore all solutions with penalty below 10,000 are feasible solutions to the original problem (solutions with penalties above 10,000 are not necessarily infeasible for the original problem though. All these instances and the best known solutions are available at http://www.cs.nott.ac.uk/~tec/NRP/ .

3 Search Neighbourhoods for Nurse Rostering

This section describes two types of search neighbourhood that have been used to solve nurse rostering problems. Their applicability to the benchmark instances are then investigated. The results from this preliminary investigation are relevant to the variable depth search presented in Section 4.

3.1 The Single Shift Neighbourhood

Included in this category are all neighbourhoods that are identified by moves or swaps that change the assignment of up to two shifts, days on/off or variables at a time. Depending on the type of cover constraints (are there minimum and/or maximum shift

cover requirements and are they hard or soft constraints?), these moves may involve either one nurse (e.g. [5, 10, 21]) or two nurses (e.g. [15, 16, 19, 27, 31, 37, 41]).

In the benchmark problems, the shift cover requirements are hard constraints and neither over nor under coverage are permitted. Therefore, once an initial feasible roster is constructed, only swaps or moves between two nurses are allowed. This ensures that the coverage constraint is not violated. Examples of these swaps are illustrated in Figure 1 and Figure 2. Figure 1, shows a section of a roster where L, N, D and DH are shifts and G, H and A are nurses. Move *a* involves the swapping of shift L and shift N on the 5th of December between nurses G and H to give the resulting roster on the right. In Figure 2, move *b* involves the assigning of shift L to nurse G from nurse H on 5th December. This could be alternatively phrased as "swapping shift L and an empty shift between nurses G and H on the 5th of December."

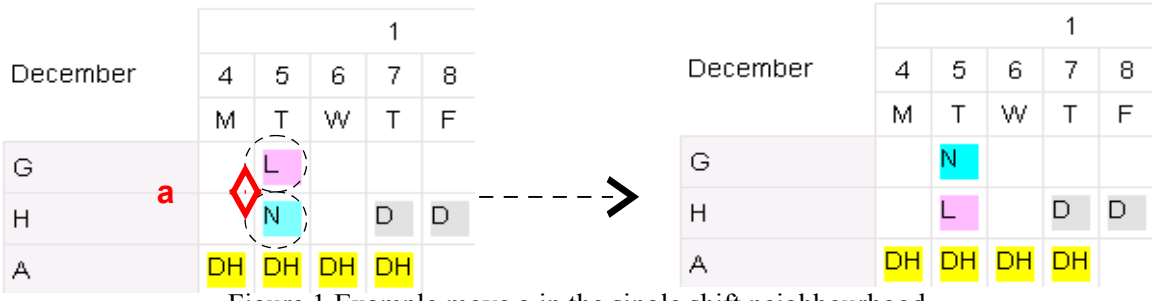

Figure 1 Example move a in the single shift neighbourhood

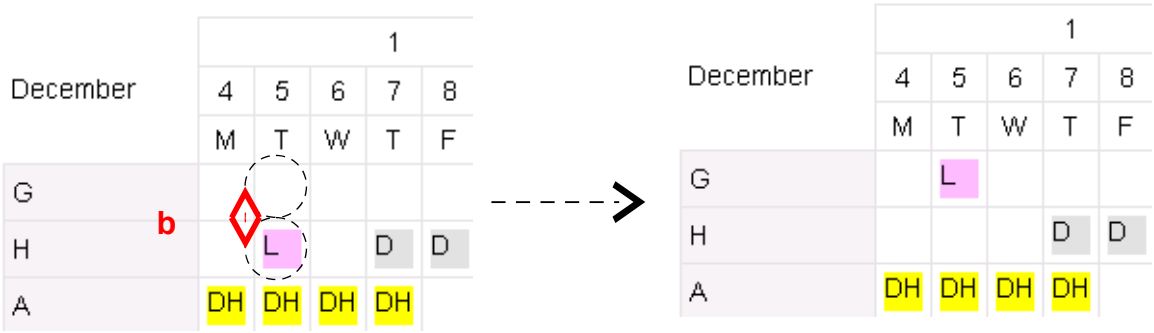

Figure 2 Example move *b* in the single shift neighbourhood

The single shift neighbourhood, is the most commonly used neighbourhood in solving nurse rostering problems. It is a relatively small neighbourhood and easy to implement in a search algorithm. Even for the largest instances examined, using today's average desktop computer, this neighbourhood can be exhaustively searched and a local optimum can be reached quickly using a hill climber in this neighbourhood. Rosters produced using this approach, however, are not always of satisfactory quality and can usually be improved by an experienced human scheduler. Therefore, this neighbourhood is often, either incorporated into a more sophisticated method such as a metaheuristic and/or replaced with a larger neighbourhood.

3.2 The Block Neighbourhood

The single shift neighbourhood, on its own, is often not effective enough. Meyer auf'm Hofe [38] highlights its weakness with a specific example in which this

neighbourhood, even if combined with a tabu list, would be unlikely to remove a particular violation as it requires the simultaneous change of eight (and only these eight) specific variables.
This section describes a larger neighbourhood defined by moves which would have been able to repair that particular violation. The neighbourhoods have recently been incorporated into search algorithms for nurse rostering and that approach has been shown to be very effective.

Included in this category of neighbourhoods are those defined by the swapping of all assignments on two or more adjacent days between two nurses. Examples of these swaps can be seen in Figure 3 and Figure 4. Figure 3 illustrates a move involving the swapping of a block of two adjacent days. On the 5th and 6th December, the N shifts of nurse A are assigned to nurse G and the L shifts of nurse G are assigned to nurse A. Here the block size is two as it involves two adjacent days. Figure 4 shows a move involving the swapping of a block of adjacent days of length three. Note that on 6th December, nurse A had no shift but this is still labelled as a swap with a block of days of length three.
As the block neighbourhood is a larger neighbourhood, its practical use was previously restricted by computational limitations. However, the recent dramatic increases in computing power have made a more aggressive use of this neighbourhood much more viable, as will be shown.

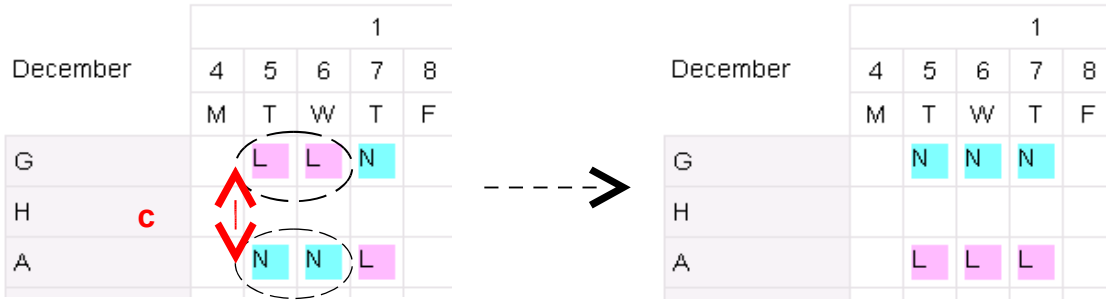

Figure 3 Example move *c* in the block neighbourhood

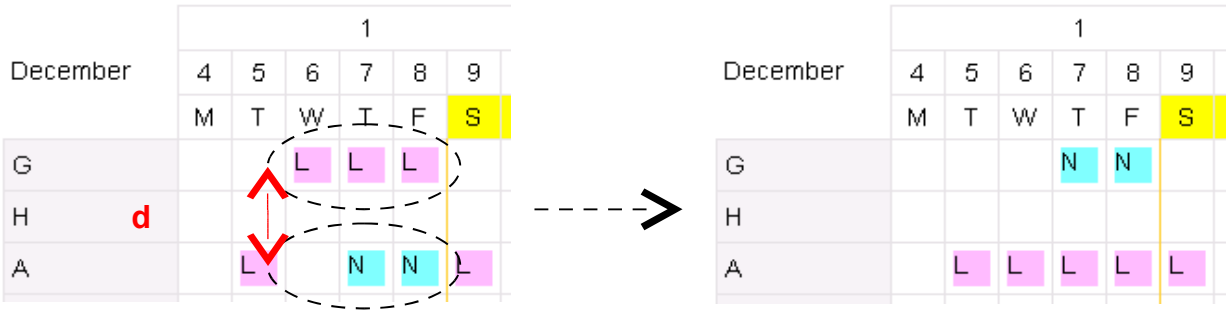

Figure 4 Example move *d* in the block neighbourhood

An early example of the use of changing blocks of shifts can be found in [29], where, although a block neighbourhood is not actually used, varying size blocks of shifts are assigned to nurses to create initial rosters. In [26], Jan et al. swap assignments on blocks of days between nurses as a mutation operator (called *escape*) in a genetic algorithm but again do not use it as a search neighbourhood. The first use of this type

of swap in a search neighbourhood can be found in [16] and subsequently [12] and [15] in which they are called *shuffle* neighbourhoods. The authors commented that although these larger neighbourhoods were very time consuming and computationally intensive to use, the solutions they produced were significantly better and almost impossible to improve by hand. A very similar neighbourhood is also used by Valouxis and Housos [45] and later, Bellanti et al. [9] also describe a neighbourhood search which partially uses similar moves.
During development and testing, it was noticed that an expert human planner will often make improvements to a roster using similar moves. This may explain why an expert has such difficulty in improving a roster produced using this search neighbourhood.

3.3 Comparing the Single Shift and Block Neighbourhood

To examine how effective these neighbourhoods are and how quickly they can be searched using today's desktop personal computer, some experiments were conducted using a simple first improvement hill climber on the benchmark instances introduced in section 2. (Note that it is actually a hill descender as the penalty is being minimised but it is referred to as a hill climber as this is, generally, a more familiar term). The search uses neighbourhoods defined by swaps of up to a maximum length block of days. The pseudocode is given in Figure 5. This pseudocode is simply to provide an outline of the process. The actual implementation contains many more lines of code that increase efficiency and avoid redundancy (e.g. not visiting a solution already examined, etc).

```
1.  WHILE there are untried swaps
2.    FOR BlockLength = 1 to MAX_BLOCK_LENGTH
3.      FOR each employee (E1) in the roster
4.        FOR each day (D1) in the planning period
5.          FOR each employee (E2) in the roster

6.            Swap all assignments between E1 and E2 on D1 up
              to D1+BlockLength

7.            IF an improvement in roster penalty THEN
8.              Break from this loop and move on to the next
                day
9.            ELSE
10.             Reverse the swap
11.           ENDIF

12.         ENDFOR
13.       ENDFOR
14.     ENDFOR
15.   ENDFOR
16. ENDWHILE
```

Figure 5 Pseudocode for the hill climber

The initial roster is created using a randomized greedy assignment method. It operates as follows: for each shift which needs to be covered, assign it to the nurse who incurs the least gain in penalty for their individual schedule (or who receives the greatest decrease in penalty) on receiving this shift.
In order to provide different starting solutions and allow the search to also be used with random restarts, the set of shifts to be assigned is randomly shuffled. The quality of the initial rosters created using this greedy algorithm is usually very poor.

As there are few hard constraints, it is not difficult to construct a feasible roster. Also, the hard constraints are all related to coverage and it is possible to pre-calculate whether a feasible solution can be built. If a feasible solution does not exist, the user is notified that the cover requirements need to be reduced or extra staff need to be added. This is important as the hill climber operates over the feasible solution space.

Table 2 presents the results of the hill climber (outlined in Figure 5) when the maximum block length parameter (MBL) is set from one to ten. Note that when MBL=1, it is effectively the single shift neighbourhood (assuming no more than one assignment per day).
Each experiment is repeated five times using different initial rosters (the same initial rosters are used for each MBL setting). The best, average and worst solutions, the average number of solutions examined per repeat and the average computation time per repeat are recorded for each instance. Table 2 contains the averages of these over all instances (for quick reference) and the results for each instance can be found in Table 3. The experiments were performed using a desktop PC with an Intel P4 2.4GHz processor.

Repeats	MBL	Best	Average	Worst	Average No. solutions examined per repeat	Average computation time per repeat (secs)
5	1	2064	2820	3576	49200	3.8
5	2	1565	2245	2614	86095	7.5
5	3	1452	1840	2224	100472	9.2
5	4	1413	1634	1893	115241	11.0
5	5	1253	1573	2005	127452	12.4
5	6	1191	1570	1891	138627	13.7
5	7	1341	1526	1829	159760	16.3
5	8	1160	1452	1652	171433	17.7
5	9	1165	1462	1712	183773	19.2
5	10	1165	1458	1721	185903	19.7
25	1	1589	2613	3888	50513	4.0

Table 2 Results of varying MAX_BLOCK_LENGTH (MBL)

Instance	Max block length	Best	Ave.	Worst	Average No. solutions examined	Average computation time (seconds)	Instance	Max block length	Best	Ave.	Worst	Average No. solutions examined	Average computation time (seconds)
ORTEC01	1	8927	14641	20192	55752	2.6	BCV-5.4.1	1	487	725	933	1560	0.1
ORTEC01	2	5917	11046	13185	83462	4.1	BCV-5.4.1	2	193	369	633	2389	0.1
ORTEC01	3	5120	7735	9771	109864	5.9	BCV-5.4.1	3	48	196	487	2433	0.1
ORTEC01	4	5020	5706	6865	126689	7.1	BCV-5.4.1	4	48	108	205	2670	0.1
ORTEC01	5	3355	5470	8045	144511	8.1	BCV-5.4.1	5	48	106	195	2680	0.1
ORTEC01	6	2745	5303	6915	167063	9.7	BCV-5.4.1	6	48	106	195	2831	0.1
ORTEC01	7	4257	4848	6470	173036	10.2	BCV-5.4.1	7	48	106	195	3032	0.1
ORTEC01	8	2475	4022	4700	185363	11.1	BCV-5.4.1	8	48	106	195	3268	0.1
ORTEC01	9	2525	4174	5295	202584	12.4	BCV-5.4.1	9	48	106	195	3452	0.2
ORTEC01	10	2525	4131	5345	188767	11.6	BCV-5.4.1	10	48	106	195	3587	0.2
BCV-1.8.1	1	328	491	698	9647	0.4	BCV-6.13.1	1	1024	1324	1597	19883	1.0
BCV-1.8.1	2	291	448	650	14206	0.6	BCV-6.13.1	2	1019	1223	1287	29736	1.6
BCV-1.8.1	3	288	332	470	20461	0.9	BCV-6.13.1	3	994	1131	1286	34222	1.9
BCV-1.8.1	4	273	330	464	22138	1.1	BCV-6.13.1	4	954	1057	1257	41324	2.4
BCV-1.8.1	5	287	338	470	22226	1.1	BCV-6.13.1	5	954	1057	1257	46422	2.8
BCV-1.8.1	6	287	336	470	25884	1.3	BCV-6.13.1	6	954	1057	1257	50864	3.1

BCV-1.8.1	7	287	337	471	30805	1.6	BCV-6.13.1	7	954	1011	1101	60173	3.8
BCV-1.8.1	8	287	337	471	33242	1.8	BCV-6.13.1	8	954	1011	1101	64769	4.2
BCV-1.8.1	9	287	329	471	35425	2.0	BCV-6.13.1	9	954	1011	1101	69157	4.5
BCV-1.8.1	10	287	327	471	37639	2.1	BCV-6.13.1	10	954	1011	1101	73355	4.9
BCV-2.46.1	1	1704	1715	1726	140592	12.4	BCV-7.10.1	1	403	555	662	10259	0.4
BCV-2.46.1	2	1618	1674	1716	246792	24.1	BCV-7.10.1	2	381	514	606	13863	0.6
BCV-2.46.1	3	1618	1663	1701	302478	31.4	BCV-7.10.1	3	381	505	596	18048	0.9
BCV-2.46.1	4	1618	1663	1701	335523	35.8	BCV-7.10.1	4	381	505	596	19851	1.1
BCV-2.46.1	5	1618	1663	1701	361902	39.5	BCV-7.10.1	5	381	505	596	21480	1.1
BCV-2.46.1	6	1618	1663	1701	387107	42.8	BCV-7.10.1	6	381	505	596	23038	1.2
BCV-2.46.1	7	1618	1663	1701	411298	46.4	BCV-7.10.1	7	381	505	596	24540	1.3
BCV-2.46.1	8	1618	1663	1701	434607	49.8	BCV-7.10.1	8	381	505	596	25982	1.4
BCV-2.46.1	9	1618	1663	1701	456895	52.9	BCV-7.10.1	9	381	505	596	27368	1.6
BCV-2.46.1	10	1618	1663	1701	478249	55.9	BCV-7.10.1	10	381	505	596	28691	1.7
BCV-3.46.1	1	3883	3969	4094	200929	16.8	BCV-8.13.1	1	236	268	334	17229	0.8
BCV-3.46.1	2	3607	3675	3801	392147	37.2	BCV-8.13.1	2	148	236	333	22756	1.1
BCV-3.46.1	3	3464	3525	3605	420806	41.7	BCV-8.13.1	3	148	198	236	28734	1.5
BCV-3.46.1	4	3474	3507	3554	483621	49.7	BCV-8.13.1	4	148	198	235	34321	1.9
BCV-3.46.1	5	3443	3463	3493	535836	56.8	BCV-8.13.1	5	148	198	235	37471	2.1
BCV-3.46.1	6	3432	3463	3486	581235	62.8	BCV-8.13.1	6	148	198	235	40455	2.3
BCV-3.46.1	7	3430	3456	3470	737642	82.7	BCV-8.13.1	7	148	198	235	43316	2.5
BCV-3.46.1	8	3408	3446	3470	800863	90.5	BCV-8.13.1	8	148	198	235	48029	2.9
BCV-3.46.1	9	3409	3446	3470	857518	98.4	BCV-8.13.1	9	148	198	235	50671	3.1
BCV-3.46.1	10	3409	3454	3507	852724	99.3	BCV-8.13.1	10	148	198	235	53218	3.3
BCV-4.13.1	1	75	110	189	12284	0.6	BCV-A.12.1	1	3570	4397	5335	23867	2.9
BCV-4.13.1	2	17	53	75	20875	1.0	BCV-A.12.1	2	2463	3210	3858	34728	4.6
BCV-4.13.1	3	22	54	75	21729	1.1	BCV-A.12.1	3	2433	3060	4015	45941	6.4
BCV-4.13.1	4	22	54	75	24019	1.3	BCV-A.12.1	4	2190	3207	3980	62257	9.2
BCV-4.13.1	5	22	52	74	31412	1.8	BCV-A.12.1	5	2275	2878	3980	70575	10.4
BCV-4.13.1	6	22	52	74	33390	2.0	BCV-A.12.1	6	2275	3019	3980	74401	11.2
BCV-4.13.1	7	22	52	74	35330	2.1	BCV-A.12.1	7	2265	3082	3980	78429	11.9
BCV-4.13.1	8	15	50	74	38808	2.4	BCV-A.12.1	8	2265	3178	3980	79397	12.3
BCV-4.13.1	9	13	49	74	45282	2.8	BCV-A.12.1	9	2265	3134	3980	89375	14.0
BCV-4.13.1	10	13	49	74	47293	3.0	BCV-A.12.1	10	2265	3134	3980	95509	15.0

Table 3 Results of varying MBL in the hill climber (5 repeats for each instance)

Instance	Max block length	Best	Ave.	Worst	Average No. solutions examined	Average computation time (seconds)
ORTEC01	1	5431	12505	21199	56621	2.6
BCV-1.8.1	1	312	432	698	9168	0.4
BCV-2.46.1	1	1634	1694	1799	158093	14.2
BCV-3.46.1	1	3694	3899	4094	199456	16.9
BCV-4.13.1	1	18	135	411	12140	0.6
BCV-5.4.1	1	194	714	1099	1676	0.1
BCV-6.13.1	1	1024	1360	1749	19574	1.0
BCV-7.10.1	1	403	535	742	9379	0.4
BCV-8.13.1	1	149	243	379	16799	0.8
BCV-A.12.1	1	3030	4609	6708	22223	2.8
Average		1589	2613	3888	50513	4.0

Table 4 Hill climber, 25 repeats with MBL=1

The results in Table 3 and Table 2 show that the single shift neighbourhood (i.e. MBL=1) is not as effective as when MBL>1. It is logical to question whether this is simply due to less solutions being examined when MBL=1. To provide a fairer comparison, the experiments were repeated for MBL=1 but with 25 instead of 5 repeats (Table 4 and the bottom row of Table 2). This ensures that the searches with MBL=1 receive at least the same time (and in most cases more) as each experiment with MBL>1. Although giving a longer computation time did improve the best solution found, it was still worse on all instances than MBL=2 and significantly worse than MBL>5.
As shown, increasing MBL increases the quality of the results but at the cost of extra computation time. However, when MBL>8, any increases in performance become less clear. Although MBL could range up to the number of days in the planning period, the results suggest that setting MBL>8 does not yield better results, especially in relation to the extra computation time required. In fact, when MBL>8 the results deteriorate slightly for some instances. This would have been a strange result if line 1 of the pseudocode was not present. However, what is happening is that a move is being made when the block length=9 that would obviously not have been made if MBL<9 and hence in the next iteration of the loop at line 1 the current solution is slightly different. It can also be seen that the increase in computation time is approximately linear in relation to MBL.

On average, increasing MBL will yield better solutions. However, if the results for each instance are studied, the benefits of using larger blocks on some instances is less noticeable. For example, on instances BCV-2.46.1 and BCV-7.10.1, MBL=2 is only slightly better than MBL=1 and increasing MBL above 3 gives no further improvement. Similarly, increasing MBL above 5 for instance BCV-5.4.1 is not worthwhile. Although the reasons for this are not obvious, it is thought to be linked to which types of constraints and their priorities are used in the schedules. Although it may be possible to estimate the suitability of neighbourhoods for a particular instance based on its constraints, it would also be difficult, as potentially each nurse could request a different set of constraint types with different parameters for any one scheduling period.

The computation times for each instance range from < 1 second to approximately 90 seconds. As would be expected, the longer computation times are required for the rosters with more employees, longer planning horizons and also those instances which utilise a larger set of the available soft constraint types for each employee (e.g instance BCV-A.12.1).

As can be seen, on the machine used, the search, on average, examines approximately 10,000 solutions per second. Examining the results for each instance reveals that the solutions examined per second ranges from approximately 28,000 for the instances with fewer soft constraints to around 6,000 for the instances which use all the available soft constraint types. Evaluating soft constraints is by far the most time consuming function in the search and so a large amount of effort was spent streamlining them to ensure that they were fast and efficient as well as accurate. It is possible to obtain large increases in search performance through writing faster code than any new heuristic or search mechanism may be able to achieve. This is not always appreciated and the challenge and importance of writing fast and efficient evaluation functions is often underestimated. Also, metrics such as the performance

per number of solutions evaluated are often analysed less, if at all. In the results, the number of rosters examined as well as computation times are provided. In the authors' opinion this is a more revealing, reliable and future-proof measure of a search method's performance.

4 The Variable Depth Search

As the results in the previous section show, using today's average desktop PC, a local search employing the block neighbourhood can be completed on the larger instances in less than 90 seconds. These solutions are very difficult to improve by hand. However, due to the complexity of the problems, they are still very often local optima (albeit high quality ones). Therefore, end users may wish to use idle computer time (e.g. during a lunch break or over night) to try and find even higher quality rosters. Perhaps the simplest way to provide this option is by restarting the hill climber as many times as possible in the allotted time with different initial rosters, in the form of a basic iterated local search [33]. This is something that was tested and the results are provided in section 5. However, the main focus of this chapter is a variable depth search which will now be introduced.

The first step of the algorithm is to create an initial roster. This is done using the greedy assignment method introduced in section 3.3. As mentioned, these initial rosters can be constructed very quickly (in less than a second) but are generally of poor quality. It was found, however, that the quality of the initial roster had relatively little impact on the final roster. Once the initial roster is created, it is possible to proceed with the variable depth search which, like the hill climber, also operates over the feasible solution space. Figure 6 provides an outline of the algorithm.
The search is similar to a method used when attempting to manually improve rosters. When improving rosters by hand it was observed that first we would try and improve one nurse's individual schedule (that is lower the penalty for that nurse's schedule). Improving this nurse's schedule would usually be at the expense of another nurse so we then try and improve their schedule. If the second nurse's schedule is improved it may be at the expense of a third nurse's schedule so we then move on to the third nurse and so on until (hopefully) we have an overall roster penalty that is lower than the original penalty. If not, we would reverse all the changes we have just made and try a different path. This is the basic idea behind the algorithm.

$penalty_r$ = the penalty for roster r.

$penalty_{r,n}$ = the penalty for the schedule of nurse n in roster r.

```
0.   set best roster    := the current roster

1.   set current roster := an unvisited neighbour in neighbourhood
                           for best roster

2.   if no unvisited neighbour available
         stop and return best roster

3.   if penalty_current roster < penalty_best roster
         goto 0.

4.   if neither of the penalties decrease for the individual schedules of
     the two employees involved in the swap OR maximum depth <= 1
         goto 1.

5.   set E1 := the employee with increased penalty
     set current depth := 1

6.   In the neighbourhood for the current roster where considering swaps
     of blocks between employee E1 and all other employees (E2)

     set current roster := neighbouring roster with lowest penalty where
     penalty_neighbour < penalty_best roster or
     penalty_neighbour - penalty_neighbour,E2 + penalty_current roster,E2
     < penalty_best roster

7.   if no such neighbour
         goto 1.
8.   else if current roster's penalty < best roster's penalty
         goto 0.
9.   else if current depth < a preset maximum depth
         set E1 := E2
         set current depth := current depth + 1;
         goto 6.
10.  else
         goto 1.
```

Figure 6 Variable depth search outline

The neighbourhoods referred to in Figure 6 are identified by swaps of blocks up to a maximum block length (MBL). The neighbourhood at step 1 is defined by all possible swaps of blocks, on all days of the planning period, between all nurses. At step 6, the swaps are just between two nurses on all days of the planning period. It was found to be generally more efficient to set MBL at step 1 lower than at step 6 (e.g at step 1, use 2 or 3 and at step 6, use 5 or 6).

At various points in the algorithm (e.g. steps 4 and 6), it is necessary to analyse the change in penalty for a nurse's individual schedule after a swap has been performed. After any swap, at least one but no more than two of the nurse's individual schedules will have been altered. However, the penalties for other nurses' schedules may also have changed even though their schedule has not been modified. This occurs in instances which use the so called 'vertical' constraints of tutorship and ensuring that certain nurses work separately. Therefore, when analysing the change in penalty for any individual nurse's schedule that has just been altered, what we actually use is the

net change in penalties for this nurse and all other nurses that are directly linked to this nurse by 'vertical' constraints.

4.1 Heuristics

It was stated that the search operates over the feasible solution space. However, it was discovered that it was beneficial to treat the hard constraint that a nurse must have the skills required to perform a shift as a soft constraint. This can be achieved by assigning a sufficiently high weight to the hard constraint violation (e.g. giving it the same value as the penalty of the initial roster) thus ensuring that a solution with this constraint violated will not be returned. Using it as a soft constraint though, allows greater exploration of the search space. This happens because rosters with this hard constraint violated are sometimes used as intermediate solutions in a chain of moves and a better local optimum may be reached via them.

Step 6 is perhaps the most important step in the algorithm. Step 6 specifies which moves to examine as potential candidates to be added to the current chain of moves and also defines the rule for deciding which one (if any) to select. As outlined in Figure 6, a swap is only selected as a potential move to add to the current chain if *ignoring the change in N2's penalty, the neighbour's penalty is less than the best roster's penalty*. This rule is similar to, and inspired by, the 'Gain Criterion' of the Lin-Kernighan algorithm for the travelling salesman problem [32]. In section 5, the results of a number of experiments are presented in which this rule is removed to investigate its benefit.

The number of moves to examine in order to select candidates for continuing the chain can have a significant effect on the performance of the algorithm. If too many moves are tested, then the algorithm's run time will increase. If too few are selected, then there is a smaller chance of a successful one being found and the algorithm will become less effective. In Figure 6, all swaps up to a maximum block length, on all days of the planning period, between one nurse and all the others are tested. Reducing the run time by limiting the number of nurses to test swaps between and reducing the number of days adjacent to the swap at step 1 over which to test swaps was evaluated. As expected, the run time was improved but at the cost of roster quality. To try and increase efficiency, two heuristics for selecting candidate moves were developed and tested instead.

In the first heuristic (**violation flag heuristic**), all days which need changing either through the removal, addition or changing of shift assignments, in order to remove a soft constraint violation are flagged during penalty recalculations. Only the swaps which involve at least one of these days are then tested. This heuristic is also applied at step 1. Only focusing on parts of a solution that have violations and need repairing is a common heuristic. For example, Nonobe and Ibaraki [40] use a similar heuristic in a tabu search approach tested on a nurse rostering problem formulated as a constraint satisfaction problem.

In the second heuristic (**worsened days heuristic**), an array of penalties due to soft constraint violations for each day is maintained for each nurse's schedule during penalty calculations. Using these arrays, the moves in step 6 are then restricted to only those blocks that contain days which were made worse (i.e. penalty increased) after

the last move. This is a more restrictive heuristic as days which contain violations will be ignored if they were not affected by the last swap in the chain.

4.2 Pre-defined Run Time

The running time for the algorithm depends on the size of the neighbourhoods at steps 1 and 6, the maximum depth used at step 9 and the structure of the instance being addressed. The size of the neighbourhoods at steps 1 and 6 depends upon the number of nurses, the number of days in the planning period and the maximum block length. The effects of the third factor (the instance structure) on the running time cannot be as easily predicted as factors such as the number of nurses and days. For some instances, it is possible that the structure (determined more by the soft constraints and their weights) is such that there is very often a valid neighbour found at step 6 with which to replace the current roster but which is not better than the best roster. This can mean that the search sometimes reaches great depths which obviously affects the running time.

To reduce this effect, a maximum depth which is set beforehand is used at step 9. Initially the depth was set using a trial and error method of running the algorithm for a short time and observing its progress on the particular instance. Then altering the maximum depth value until a suitable setting is found (that is estimated) will restrict the algorithm to a satisfactory running time. This is obviously not a suitable approach for practical use. Therefore, an additional mechanism was added which takes the preferred running time as a parameter and attempts to use that time efficiently.

This mechanism works as follows: for the algorithm to finish, every neighbour in the neighbourhood at step 1 needs to be examined and potentially used as the first solution in a chain of moves. It is possible to calculate the size of the neighbourhood at step one using the number of nurses, the maximum block length and the number of days in the planning horizon. Given a preferred running time and the number of solutions to evaluate at step one (updated each time a new best solution is found), it is possible to calculate an average time to spend using each neighbour at step one as the first solution in a chain. Then at step 9, instead of testing whether a maximum depth will be exceeded in continuing the chain, we test whether the average time per chain will be exceeded if it continues.

In the results section where this heuristic is not used but a maximum running time is set, the search immediately terminates and returns the best solution when the time limit is reached. If the algorithm naturally terminates and the preferred running time has not been exceeded, then at step 2, instead of returning the best roster, a new initial roster is created and the algorithm restarts at step 0.

Figure 7 shows an example of an improving chain of moves. The change in the roster consists of seven moves which, when performed simultaneously, provide an overall reduction in the roster's penalty. It can be seen that the second nurse of a swap is always the first nurse in the next swap. Note that Figure 7 is just used to illustrate the idea of a chain of moves. For the roster shown (which is far from optimal), there are many other chains which would also improve the roster.

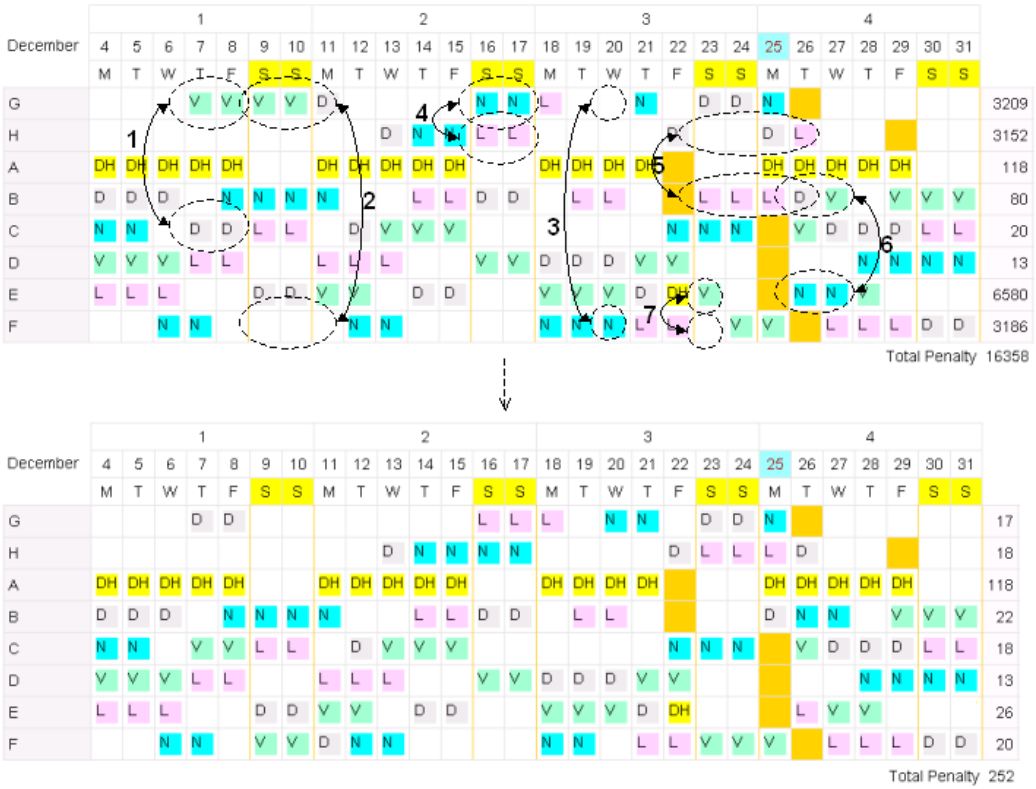

Figure 7 Example chains of swaps

From observing the logs after searches, the lengths of improving chains of moves varies greatly. Note that poor rosters are easily improved by single moves. When the local optima start to be found less frequently, (and the penalty is approaching better values) even for the smaller instances, improving chains of moves with lengths over one hundred are not uncommon.

4.3 Efficient Implementation

At step 6, there is a possibility that a neighbouring solution will be selected that has been visited previously and cycling could occur. Two different methods were tested to remove this risk. In the first method, a history of solutions visited along the current path (i.e. since the last use of step 1) is maintained and used to ensure that they are not revisited. This is fast and prevents cycling but does not guarantee that solutions are not revisited at other points in the algorithm, for example visiting a solution at step 1 that has already been used at step 6. The second method maintains a hash table of all solutions visited during the run of the algorithm and so also prevents all solutions being revisited. Testing showed that the first, simpler method, produced better results. This appears to be because the probability of visiting a duplicate solution at the points which the first method does not prevent is small and much lower than the probability of cycling at step 6. Therefore, using the faster method which prevents the majority of cycling and revisiting duplicate solutions was more efficient than the slower approach which guaranteed no cycling or duplicate paths.

As discussed earlier, increases in performance can be achieved as effectively through making the algorithm faster and more efficient as by using better heuristics. We have already mentioned the importance of avoiding cycling. There are also some other

efficiency measures which are worth highlighting. Firstly, when a nurse's schedule has been altered it is only necessary to re-evaluate their schedule and any other nurses' schedules which may be linked by vertical constraints to recalculate the new roster's penalty. Secondly, by far the most time consuming operation is calculating penalties (i.e. soft constraint evaluations). If there is a likelihood that a solution will be returned to, then the algorithm caches penalties to avoid having to recalculate them. Finally, some soft constraint calculations can be speeded up by using data structures that are modified as assignments are made. A simple example is to update the total number of hours worked when a shift is (un)assigned rather than to add all the hours up when calculating the penalty. Some of the more complicated constraints benefit from a similar approach.

5 Results

The algorithm was tested on the ten publicly available data sets introduced in section 2.

5.1 Comparing Heuristics

In the experiments, the following parameters were used: Maximum depth = 1000, maximum block length at step 1 = 2, maximum block length at step 6 = 5. As described, if the algorithm finishes before the maximum running time is reached, a new initial solution is constructed and the search restarts.
Table 5, Table 6, Table 7 and Table 8 contain the results of the variable depth search when different heuristics are used. Predefined maximum run times of 1 minute, 2 minutes and 5 minutes were tested with each instance and repeated five times using different random seeds (but which were the same for each corresponding trial). The best, worst and average of these runs were recorded. The penalties in Table 5 are the averages over all instances. The results for each instance given in Table 6, Table 7 and Table 8.

The heuristics are summarised below:

No heuristics Step 4 is removed and step 5 is applied twice but with E1 set as the other nurse each time. At step 6 the only rule is to select the best neighbour.

PG Partial gain heuristic (only add a swap to the end of the current chain if it satisfies the partial improvement criterion).

VF Violation flag heuristic (only test swaps which include at least one day which is flagged as having a violation). This is used at steps 1 and 6.

WD Only swap worsened days heuristic (only test swaps which include at least one day which has a violation and the violation's penalty was increased or the violation occurred after the last swap). This is used at step 6.

TR Time restriction heuristic (ensure that a pre-calculated average amount of time is spent on every chain).

ILS Hill climber with restarts. Maximum depth is set to 1 and maximum block size at step 1 is increased to 5.

	1 Minute			2 Minutes			5 Minutes		
Heuristics	Best	Avg.	Worst	Best	Avg.	Worst	Best	Avg.	Worst
No heuristics	1390	1561	1879	1196	1454	1825	1154	1394	1766

PG	917	1156	1461	888	1078	1271	872	1034	1225
VF	1222	1629	2050	1113	1329	1611	973	1111	1232
WD	1046	1241	1515	1008	1131	1251	972	1087	1234
PG + VF	1003	1228	1680	894	1163	1619	872	1016	1303
PG + WD	900	1016	1205	895	973	1083	877	946	1031
TR	1153	1454	1829	1175	1450	1654	1008	1232	1386
TR + PG + VF	1014	1122	1277	888	994	1141	874	978	1141
TR + PG + WD	899	1068	1260	862	935	1027	858	889	938
ILS	1164	1266	1405	1048	1174	1290	1012	1067	1161

Table 5 Comparison of heuristic combinations

Time restriction = 1 minute. Max. depth = 1000. Max. block length at step 6 = 5. Max. block length at step 1 = 2.

		No heuristics	PG	VF	WD	PG + VF	PG + WD	TR	TR + PG + VF	TR + PG + WD	ILS
ORTEC01	Average	5768	2358	6560	3234	3347	1103	4605	2242	1910	3573
	Best	4826	506	3245	1655	1421	425	2746	1430	555	2825
BCV-1.8.1	Average	263	263	267	265	263	259	278	257	261	266
	Best	255	253	259	257	254	254	271	254	255	261
BCV-2.46.1	Average	1679	1667	1643	1662	1659	1683	1657	1605	1598	1641
	Best	1653	1616	1595	1633	1616	1646	1618	1593	1574	1618
BCV-3.46.1	Average	3570	3478	3546	3515	3501	3481	3588	3477	3411	3464
	Best	3483	3411	3456	3505	3460	3422	3531	3442	3380	3443
BCV-4.13.1	Average	11	13	10	11	11	10	22	11	11	12
	Best	10	10	10	10	10	10	11	10	10	11
BCV-5.4.1	Average	48	48	48	48	48	48	48	48	48	48
	Best	48	48	48	48	48	48	48	48	48	48
BCV-6.13.1	Average	941	873	793	770	814	769	844	768	770	908
	Best	827	768	768	768	768	768	769	768	768	822
BCV-7.10.1	Average	419	455	385	384	383	384	392	384	384	381
	Best	381	381	381	381	381	381	381	381	381	381
BCV-8.13.1	Average	180	164	148	148	148	148	196	148	148	148
	Best	148	148	148	148	148	148	148	148	148	148
BCV-A.12.1	Average	2731	2240	2890	2370	2103	2271	2912	2284	2135	2215
	Best	2265	2033	2309	2050	1919	1900	2009	2068	1870	2085

Table 6 Variable depth search heuristics with max. run time 1 minute

Time restriction = 2 minutes. Max depth = 1000. Max. block length at step 6 = 5. Max. block length at step 1 = 2.

		No heuristics	PG	VF	WD	PG + VF	PG + WD	TR	TR + PG + VF	TR + PG + WD	ILS
ORTEC01	Average	5023	1931	4246	2529	2893	860	4990	1262	868	2713
	Best	3110	485	2450	1605	520	425	3099	470	480	1691
BCV-1.8.1	Average	263	261	265	262	260	255	269	257	257	265
	Best	255	252	257	257	254	253	263	253	253	261
BCV-2.46.1	Average	1679	1664	1642	1652	1649	1680	1627	1597	1600	1620
	Best	1653	1616	1595	1633	1596	1646	1616	1572	1574	1615
BCV-3.46.1	Average	3486	3433	3477	3433	3477	3457	3523	3410	3380	3446
	Best	3449	3392	3419	3414	3399	3372	3479	3380	3338	3413
BCV-4.13.1	Average	11	11	10	11	10	10	11	11	10	12
	Best	10	10	10	10	10	10	10	10	10	11

BCV-5.4.1	Average	48	48	48	48	48	48	48	48	48	48
	Best	48	48	48	48	48	48	48	48	48	48
BCV-6.13.1	Average	904	842	780	769	768	769	836	768	768	889
	Best	827	768	768	768	768	768	774	768	768	822
BCV-7.10.1	Average	419	437	384	384	382	384	403	384	384	381
	Best	381	381	381	381	381	381	381	381	381	381
BCV-8.13.1	Average	164	148	148	148	148	148	148	148	148	148
	Best	148	148	148	148	148	148	148	148	148	148
BCV-A.12.1	Average	2540	2007	2293	2071	1992	2115	2647	2051	1882	2213
	Best	2075	1775	2050	1819	1820	1900	1928	1845	1620	2085

Table 7 Variable depth search heuristics with max. run time 2 minutes

Time restriction = 5 minutes. Max depth = 1000. Max. block length at step 6 = 5. Max. block length at step 1 = 2.

		No heuristics	PG	VF	WD	PG + VF	PG + WD	TR	TR + PG + VF	TR + PG + WD	ILS
ORTEC01	Average	4763	1785	2421	2234	1570	857	3150	1147	448	1901
	Best	3000	460	1390	1410	435	425	1420	360	420	1590
BCV-1.8.1	Average	259	258	262	255	255	254	264	257	254	263
	Best	254	252	256	253	253	253	255	253	253	257
BCV-2.46.1	Average	1664	1654	1621	1635	1647	1669	1622	1596	1586	1607
	Best	1633	1614	1595	1594	1595	1613	1595	1572	1572	1595
BCV-3.46.1	Average	3468	3404	3465	3392	3448	3423	3443	3392	3378	3427
	Best	3427	3374	3411	3382	3399	3372	3399	3347	3355	3413
BCV-4.13.1	Average	11	11	10	10	10	10	12	10	10	11
	Best	10	10	10	10	10	10	10	10	10	10
BCV-5.4.1	Average	48	48	48	48	48	48	48	48	48	48
	Best	48	48	48	48	48	48	48	48	48	48
BCV-6.13.1	Average	857	818	768	769	768	768	814	768	768	853
	Best	768	768	768	768	768	768	770	768	768	777
BCV-7.10.1	Average	401	401	382	383	382	383	384	383	383	381
	Best	381	381	381	381	381	381	381	381	381	381
BCV-8.13.1	Average	148	148	148	148	148	148	148	148	148	148
	Best	148	148	148	148	148	148	148	148	148	148
BCV-A.12.1	Average	2320	1814	1985	2000	1881	1900	2437	2028	1867	2034
	Best	1869	1664	1720	1729	1679	1750	2053	1849	1620	1900

Table 8 Variable depth search heuristics with max. run time 5 minutes

The results indicate that the best combination of heuristics is TR+PG+WD. The better results all use PG. TR is improved if combined with other heuristics and similarly VF and WD also perform better when not used on their own. As would be expected, VF and WD perform similarly (WD is a more restrictive form of VF). ILS performs similarly to TR, VF and WD when they are used on their own. All other combinations outperform ILS.

Examining the results for each instance is also interesting. VF is very effective on instance BCV-4.13.1, even when used on its own and WD+PG works well on instance ORTEC01. However, we were unable to draw any definite conclusions on why this should be the case after examining the characteristics of these instances more closely. Due to its simplicity ILS is able to examine more solutions in the allocated time but its lack of heuristics makes it less effective.

5.2 Comparisons with Other Methods

Brucker et al. [11] developed a heuristic constructive approach and tested it on the benchmark instances. As it is a constructive method it is not possible to provide a comparison to the variable depth search by using the number of solutions examined metric. However, their experiments were performed on the same machine and a comparison can be provided by using computation times. The results in Table 9 are Brucker et al's best results from all experiments. The total computation time in obtaining these solutions for each instance was then set as the maximum run time for the variable depth search with the heuristic combination TR+PG+VF.

Burke et al's result for ORTEC01 using the hybrid variable neighbourhood search [13] had a computation time of 12 hours. The result for the variable depth search on this instance is the best of the five, five minute tests using the heuristic combination TR+PG+VF.

As can be seen, the variable depth search outperforms the constructive method, over the same computation times, on all instances except one, on which they are equal. It also beats the hybrid method of Burke et al. on instance ORTEC01.

	Time	Brucker et al. [11]	Burke et al. [13]	Variable Depth Search
ORTEC01	12 hrs	-	541	360
BCV-1.8.1	136 sec	323	-	253
BCV-2.46.1	3424 sec	1594	-	1572
BCV-3.46.1	2888 sec	3601	-	3324
BCV-4.13.1	208 sec	18	-	10
BCV-5.4.1	16 sec	200	-	48
BCV-6.13.1	304 sec	890	-	768
BCV-7.10.1	216 sec	396	-	381
BCV-8.13.1	224 sec	148	-	148
BCV-A.12.1	944 sec	3335	-	1843

Table 9 Comparison of VDS with other algorithms

To provide further comparisons, the hybrid tabu search of Burke et al. [12, 16] was implemented and tested on the benchmark data sets. The best version of their tabu search (TS2) was applied five times to each instance. Table 10 contains the best and average results. The average execution time on each instance was also recorded. The variable depth search was then set a maximum run time identical to that used by the tabu search for each instance. Five repeats of the variable depth search were then performed to obtain average and best results. Heuristics TR+PG+WD were used.

	Variable depth search			TS2 [12, 16]			
Instance	Best	Average	Avg. Evals.	Best	Average	Avg. Evals.	Time (secs)
ORTEC01	480	1120	1,852,788	1581	3201	2,363,828	108
BCV-1.8.1	262	269	159,379	293	350	140,690	9
BCV-2.46.1	1574	1593	1,563,865	1573	1596	1,557,905	167
BCV-3.46.1	3334	3346	4,486,399	3410	3453	5,088,206	427
BCV-4.13.1	10	11	152,182	11	25	150,639	9
BCV-5.4.1	48	48	21,208	48	48	17,580	1
BCV-6.13.1	769	817	356,891	1010	1154	345,595	24
BCV-7.10.1	381	427	117,224	391	458	93,817	7
BCV-8.13.1	148	148	248,927	148	165	215,524	15
BCV-A.12.1	1835	1942	649,926	2065	2831	718,354	108

Table 10 Comparison of the variable depth search with a hybrid tabu search

The results in Table 9 and Table 10 show the variable depth search nearly always outperforms or is equal to previous methods in comparable tests over all instances. The only time it was beaten was when TS2 found a best solution for BCV-2.46.1 with penalty 1573. The variable depth search could still manage a best with penalty 1574 though. Note that the variable depth search is actually dynamically adjusting to the run time of the other approaches.

5.3 Longer Computation Times

Further experiments were conducted to examine the potential benefit of a longer execution time. The variable depth search with the same parameters as before and using heuristics TR+PG+WD was tested on each instance with a time limit of 1 hour. ILS was also tested for comparative purposes.

	TR+PG+WD		ILS	
	Pen	Evals	Pen	Evals
ORTEC01	435	58,703,568	630	69,151,556
BCV-1.8.1	254	62,769,622	255	72,216,494
BCV-2.46.1	1574	29,767,138	1594	39,255,535
BCV-3.46.1	3302	36,422,606	3414	42,504,506
BCV-4.13.1	10	58,613,414	10	68,301,984
BCV-5.4.1	48	73,021,195	48	92,071,328
BCV-6.13.1	768	54,683,305	814	62,509,316
BCV-7.10.1	381	59,704,319	381	69,957,265
BCV-8.13.1	148	59,084,678	148	66,960,140
BCV-A.12.1	1564	23,675,881	1808	25,506,048
Average	848		910	

Table 11 Experiments with VDS using longer computation times

The increase in computation time leads to an improvement for both methods. Again the simpler ILS examines more solutions in the allotted time but is still not as effective as the variable depth search. In fact, ILS after one hour on each instance is, on average, still worse than the best of five, one minute repeats of the variable depth search with heuristics PG + WD.

6 Conclusions

This paper has reviewed search neighbourhoods that have been previously used to solve nurse rostering problems. They were tested using benchmark nurse rostering problems and based on the results, a variable depth search was created.

The block neighbourhood is very effective for the majority of the instances. Today's technology allows these larger neighbourhoods to be exhaustively searched very quickly. Even a simple hill climber which uses these neighbourhoods will produce satisfactory rosters and combining the hill climber with the greedy construction, restart method further improves the quality of solutions produced.

However, the variable depth search can still improve upon this basic iterated local search. The variable depth search works by chaining together the block moves using a number of heuristics to select the next move (link in the chain). The results show the best combination of heuristics to use is PG (positive gain criterion) with TR (time restriction heuristic) and WD (selecting moves on days that have violations which occurred after the last move).

It is also worth noting that although the variable depth search is more effective, it is also more complicated to implement with an increased potential for introducing errors.

7 Suggestions for Future Work

It is possible that the iterated local search can be improved by using more sophisticated heuristics for introducing diversification than the greedy re-construction method. The hill climber using the block neighbourhoods or the variable depth search could also be used in a population based approach such as a memetic algorithm [30] or a scatter search [25] as the means for introducing diversification.
Specific neighbourhoods (i.e. maximum block size settings) and heuristics are particularly effective on certain instances. A method which can exploit this by, for example, intelligently selecting neighbourhoods and heuristics, may be able to contribute gains in performance. One possibility may be an algorithm which runs some short preliminary tests on the instance, testing different parameters and heuristics in order to estimate a good set of parameters for the variable depth search. An alternative approach may be to dynamically adjust the algorithm's set of heuristics and parameters as the search progresses. Somewhat akin to hyperheuristics [18, 43].
We also plan to create more data sets taken from real world rostering scenarios which along with any new best known solutions will be published on the research website.

Acknowledgements

This work was supported by EPSRC grant GR/S31150/01.

References

1. Ahuja, R.K., Ö. Ergun, J.B. Orlin, and A.P. Punnen, *A survey of very large-scale neighborhood search techniques.* Discrete Applied Mathematics, 2002. **123**(1-3): pp. 75-102.
2. Aickelin, U., E.K. Burke, and J. Li, *An Estimation of Distribution Algorithm with Intelligent Local Search for Rule-based Nurse Rostering.* Journal of the Operational Research Society, 2007(in print).
3. Arthur, J.L. and A. Ravindran, *A multiple objective nurse scheduling model.* AIIE Transactions, 1981. **13**(1): pp. 55-60.
4. Azaiez, M.N. and S.S. Al Sharif, *A 0-1 goal programming model for nurse scheduling.* Computers and Operations Research, 2005. **32**(3): pp. 491 - 507.
5. Bard, J.F. and H.W. Purnomo, *Preference scheduling for nurses using column generation.* European Journal of Operational Research, 2005. **164**(2): pp. 510-534.
6. Bard, J.F. and H.W. Purnomo, *Cyclic Preference Scheduling of Nurses Using A Lagrangian-Based Heuristic.* Journal of Scheduling, 2007. **10**(1): pp. 5-23.
7. Beddoe, G.R. and S. Petrovic, *Selecting and Weighting Features Using a Genetic Algorithm in a Case-Based Reasoning Approach to Personnel Rostering.* European Journal of Operational Research, 2006. **175**(2): pp. 649-671.
8. Beddoe, G.R. and S. Petrovic, *Enhancing Case-Based Reasoning for Personnel Rostering with Selected Tabu Search Concepts.* Journal of the Operational Research Society, To appear.

9. Bellanti, F., G. Carello, F.D. Croce, and R. Tadei, *A greedy-based neighborhood search approach to a nurse rostering problem.* European Journal of Operational Research, 2004. **153**: pp. 28–40.
10. Berrada, I., J.A. Ferland, and P. Michelon, *A multi-objective approach to nurse scheduling with both hard and soft constraints.* Socio-Economic Planning Sciences, 1996. **30**(3): pp. 183-193.
11. Brucker, P., E.K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe, *Adaptive Construction of Nurse Schedules: A Shift Sequence Based Approach and New Benchmarks.* Under review, 2006.
12. Burke, E.K., P. Cowling, P. De Causmaecker, and G. Vanden Berghe, *A Memetic Approach to the Nurse Rostering Problem.* Applied Intelligence, 2001. **15**(3): pp. 199-214.
13. Burke, E.K., T. Curtois, G. Post, R. Qu, and B. Veltman, *A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem.* European Journal of Operational Research, Accepted for publication, to Appear 2007.
14. Burke, E.K., T. Curtois, G. Post, R. Qu, and B. Veltman, *A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem.* European Journal of Operational Research, To Appear.
15. Burke, E.K., P. De Causmaecker, S. Petrovic, and G. Vanden Berghe, *Variable Neighborhood Search for Nurse Rostering Problems*, in *Metaheuristics: Computer Decision-Making*, M.G.C. Resende and J.P. de Sousa, Editors. 2004, Kluwer. pp. 153-172.
16. Burke, E.K., P. De Causmaecker, and G. Vanden Berghe. *A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem*, in *Simulated Evolution and Learning, Selected Papers from the 2nd Asia-Pacific Conference on Simulated Evolution and Learning, SEAL 98, Springer Lecture Notes in Artificial Intelligence Volume 1585*. B. McKay, et al., Editors. 1999: Springer. pp. 187-194.
17. Burke, E.K., P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem, *The State of the Art of Nurse Rostering.* Journal of Scheduling, 2004. **7**(6): pp. 441 - 499.
18. Burke, E.K., G. Kendall, and E. Soubeiga, *A Tabu-Search Hyperheuristic for Timetabling and Rostering.* Journal of Heuristics, 2003. **9**(6): pp. 451 - 470.
19. Chiarandini, M., A. Schaerf, and F. Tiozzo. *Solving Employee Timetabling Problems with Flexible Workload using Tabu Search*, in *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT-2000)*. E.K. Burke and W. Erben, Editors. 2000. Konstanz, Germany. pp. 298-302.
20. Darmoni, S.J., A. Fajner, N. Mahé, A. Leforestier, M. Vondracek, O. Stelian, and M. Baldenweck, *Horoplan: computer-assisted nurse scheduling using constraint-based programming* Journal of the Society for Health Systems, 1995. **5**: pp. 41-54.
21. Dias, T.M., D.F. Ferber, C.C. de Souza, and A.V. Moura, *Constructing nurse schedules at large hospitals.* International Transactions in Operational Research, 2003. **10**(3): pp. 245-265.
22. Dowsland, K.A., *Nurse scheduling with tabu search and strategic oscillation.* European Journal of Operational Research, 1998. **106**(2): pp. 393-407.

23. Ernst, A.T., H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier, *An Annotated Bibliography of Personnel Scheduling and Rostering.* Annals of Operations Research, 2004. **127**: pp. 21–144.
24. Glover, F., *Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems.* Discrete Applied Mathematics, 1996. **65**(1-3): pp. 223-253.
25. Glover, F., M. Laguna, and R. Martí, *Fundamentals of scatter search and path relinking.* Control and Cybernetics, 2000. **29**(3): pp. 653-684.
26. Jan, A., M. Yamamoto, and A. Ohuchi. *Evolutionary Algorithms for Nurse Scheduling Problem*, in *Proceedings of the 2000 Congress on Evolutionary Computation*. 2000. California, USA: IEEE Press. pp. 196-203.
27. Jaszkiewicz, A., *A metaheuristic approach to multiple objective nurse scheduling.* Foundations of Computing and Decision Sciences, 1997. **22**(3): pp. 169-183.
28. Jaumard, B., F. Semet, and T. Vovor, *A Generalized Linear Programming Model for Nurse Scheduling.* European Journal of Operational Research 1998. **107**(1): pp. 1-18.
29. Kostreva, M.M. and K.S.B. Jennings, *Nurse scheduling on a microcomputer.* Computers and Operations Research, 1991. **18**(9): pp. 731-739.
30. Krasnogor, N. and J. Smith, *A Tutorial for Competent Memetic Algorithms: Model, Taxonomy and Design Issues.* IEEE Transactions on Evolutionary Computation, 2005. **9**(5): pp. 474-488.
31. Li, H., A. Lim, and B. Rodrigues. *A Hybrid AI Approach for Nurse Rostering Problem*, in *Proceedings of the 2003 ACM symposium on Applied computing*. 2003. pp. 730-735.
32. Lin, S. and B.W. Kernighan, *An Effective Heuristic Algorithm for the Traveling-Salesman Problem.* Operations Research, 1973. **21**(2): pp. 498-516.
33. Lourenço, H.R., O.C. Martin, and T. Stützle, *Iterated Local Search*, in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Editors. 2003, Kluwer. pp. 321-353.
34. Louw, M.J., I. Nieuwoudt, and J.H. van Vuuren, *Finding Good Nursing Duty Schedules: A Case Study.*
35. Mason, A.J. and M.C. Smith. *A Nested Column Generator for solving Rostering Problems with Integer Programming*, in *International Conference on Optimisation: Techniques and Applications*. L. Caccetta, et al., Editors. 1998. Perth, Australia. pp. 827-834.
36. Meisels, A., E. Gudes, and G. Solotorevsky, *Employee Timetabling, Constraint Networks and Knowledge-Based Rules: A Mixed Approach*, in *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 1154*, E. Burke and P. Ross, Editors. 1995. pp. 93-105.
37. Meisels, A. and A. Schaerf, *Modelling and solving employee timetabling problems.* Annals of Mathematics and Artificial Intelligence, 2003. **39**: pp. 41-59.
38. Meyer auf'm Hofe, H., *Solving Rostering Tasks as Constraint Optimization*, in *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 2079* E. Burke and W. Erben, Editors. 2000. pp. 191-212.
39. Miller, H.E., W.P. Pierskalla, and G.I. Rath, *Nurse Scheduling Using Mathematical Programming.* Operations Research, 1976. **24**(5): pp. 857-870.

40. Nonobe, K. and T. Ibaraki, *A tabu search approach to the constraint satisfaction problem as a general problem solver.* European Journal of Operational Research, 1998. **106**(2-3): pp. 599-623.
41. Petrovic, S., G.R. Beddoe, and G. Vanden Berghe. *Storing and adapting repair experiences in employee rostering*, in *Selected Papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Springer Lecture Notes in Computer Science Volume 2740*. E.K. Burke and P. De Causmaecker, Editors. 2003. pp. 149-166.
42. Rego, C., *A Subpath Ejection Method for the Vehicle Routing Problem.* Management Science, 1998. **44**(10): pp. 1447-1459.
43. Ross, P., *Hyper-heuristics*, in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E.K. Burke and G. Kendall, Editors. 2005, Springer. pp. 529-556.
44. Thornton, J. and A. Sattar. *Nurse Rostering and Integer Programming Revisited*, in *International Conference on Computational Intelligence and Multimedia Applications*. B. Verma and X. Yao, Editors. 1997. pp. 49-58.
45. Valouxis, C. and E. Housos, *Hybrid optimization techniques for the workshift and rest assignment of nursing personnel.* Artificial Intelligence in Medicine, 2000. **20**: pp. 155-175.
46. Vanden Berghe, G. *An Advanced Model and Novel Meta-Heuristic Solution Methods to Personnel Scheduling in Healthcare*. Ph.D. Thesis, University of Gent, Belgium, 2002.
47. Warner, D.M., *Scheduling Nursing Personnel according to Nursing Preference: A Mathematical Programming Approach* Operations Research, 1976. **24**: pp. 842-856.
48. Yagiura, M., T. Yamaguchi, and T. Ibaraki, *A Variable Depth Search Algorithm for the Generalized Assignment Problem*, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, et al., Editors. 1999, Kluwer Academic Publishers: Boston, MA. pp. 459-471.