# A Scatter Search Approach to the Nurse Rostering Problem

**Edmund K. Burke[1], Timothy Curtois[1], Rong Qu[1], Greet Vanden Berghe[2,3]**

[1]School of Computer Science, 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
[3]K.U.Leuven, Department of Computer Science, E. Sabbelaan 53, 8500 Kortrijk, Belgium

**Abstract**

The benefits of automating the nurse scheduling process in hospitals include reducing the planning workload and associated costs and being able to create higher quality and more flexible schedules. This has become more important recently in order to retain nurses and to attract more people into the profession. Better quality rosters also reduce fatigue and stress due to overwork and poor scheduling and help to maximise the use of leisure time by satisfying more requests. A more contented workforce will lead to higher productivity, increased quality of patient service and a better level of healthcare.

This paper presents a scatter search approach for the problem of automatically creating nurse rosters. Scatter search is an evolutionary algorithm which has been successfully applied across a number of problem domains. To adapt and apply scatter search to nurse rostering, it was necessary to develop novel implementations of some of scatter search's subroutines. The algorithm was then tested on publicly available real world benchmark instances and compared against previously published approaches. The results show the proposed algorithm is a robust and effective method on a wide variety of real world instances.

## 1 Introduction

This paper presents a scatter search approach to the nurse rostering problem. Like genetic algorithms, memetic algorithms, particle swarm optimisation and ant colony optimisation, a key feature of scatter search is the maintenance of a population of solutions. This is in contrast to many other metaheuristics which generally work with one solution, for example simulated annealing, tabu search, and variable neighbourhood search. In genetic algorithms, for example, new solutions are generally created from two parent solutions in the population through crossover and mutation operations. Although, for different problems, the details of the crossover and mutation functions can vary, there is typically some stochastic element to their operation. This contrasts with scatter search in which the method for forming new solutions is designed to minimize (if not eliminate) decisions being allocated to random (or more usually pseudo-random) chance. The idea is to try and replace calls to the random

number function with "*systematic and strategically designed rules*" (Glover et al., 2000). Another difference is found in the way that new solutions are added to the population or reference set. In many genetic algorithms, new solutions are allowed to enter the current population if their quality (usually determined by an objective function) is greater than the worst member of the current population. In scatter search, a method for comparing the similarity of two solutions is used to measure the reference set's overall diversity. Whether or not a new solution enters the reference set may then be decided by not only its quality, but also its contribution to the reference set's diversity.

Some genetic algorithms also use a local search or other optimisation method on each of the new solutions between generations in order to improve their quality. These methods may also act to repair the solutions if they were incomplete or infeasible after the crossover stage. These genetic algorithms (with local search) are often labelled as memetic algorithms but may also be referred to as hybrid genetic algorithms or genetic local search. The idea of using a heuristic improvement process on new solutions is also common to scatter search. Although these improvers/repairers can significantly improve the solutions they can also cause longer execution times for the algorithm. Also, as new solutions can be created from more than two reference solutions (in contrast to genetic algorithms), even with a small reference set there are many potential groups of parents, and so many new solutions can be created at each iteration. Therefore, the reference set is typically a lot smaller than the corresponding population in a genetic algorithm.

However, as the boundaries between metaheuristic algorithm classification sometimes overlap and as different metaheuristic approaches are often hybridised, so also, features of scatter search may appear in genetic algorithms and vice versa. The comparisons between genetic algorithms and scatter search described here are just a basic introduction. For further information on scatter search and a more in depth analysis see (Glover, 1998; Glover et al., 2000; Glover et al., 2003; Laguna and Martí, 2003).

At the time the work presented here was undertaken, there had been very little research into investigating scatter search for personnel scheduling and no known applications of it to nurse rostering. This made it an appealing method to investigate, especially considering how successful evolutionary approaches for nurse rostering have been previously. This is a conclusion which was simultaneously but independently made by Maenhout and Vanhoucke. They have also implemented and tested a scatter search for the nurse rostering problem

(Maenhout and Vanhoucke, 2006). Interestingly, their implementation of Glover's original template is very different to ours. However, they also were able to achieve successful results albeit on a variation of the nurse rostering problem. Scatter search and path relinking strategies have been applied to a considerable variety of problems other than nurse scheduling though. For example, arc routing (Greistorfer, 2003), linear ordering (Campos et al., 2001), quadratic assignment (Cung et al., 1997), mixed integer programming (Glover et al., 2000) and exam proctor assignment (Lourenço et al., 2000). All of these studies have demonstrated promising results.

In the next section we introduce the nurse rostering problem. Section 3 describes the scatter search implementation and section 4 contains the results of testing this algorithm on a variety of benchmark nurse rostering problems. To help draw conclusions the scatter search has been compared against Brucker et al.'s constructive method (Brucker et al., 2009) and the memetic algorithm of Burke et al. (Burke et al., 2001). Finally we conclude in section 5 with some views on the success of this research.

First though, we briefly discuss some of the population based optimisation methods that have previously and successfully been applied to employee timetabling problems in various forms.

Jan et al. (Jan et al., 2000) evaluate the use of a genetic algorithm to solve a nurse scheduling problem. Although the authors note that the problem is simplified slightly as they are conducting preliminary tests, some common constraints (coverage and personal requests) and objectives (mostly related to night shifts) are still present. The authors also suggest a method for allowing a decision maker to adjust the schedule and guide the search during its execution.

Aickelin and Dowsland (Aickelin and Dowsland, 2000) developed and tested a genetic algorithm in place of the tabu search method in (Dowsland and Thompson, 2000). They were able to achieve a similar performance with the genetic algorithm and felt it was more robust when applied to a greater variety of instances. In (Aickelin and Dowsland, 2003), the same authors also tested an indirect genetic algorithm on the same problem. This time, the genetic algorithm is used to identify permutations of nurses which are then passed to a decoder which applies heuristic rules to this permutation to assign work patterns and to construct the roster. After fine tuning the heuristics and objective weights, the algorithm was capable of slightly better results than the direct genetic algorithm.

Burke et al. (Burke et al., 2001) present a number of memetic algorithms for nurse scheduling. Experiments are conducted combining different crossover operators and local

improvements methods. The best approach is a hybridisation of a tabu search (Burke et al., 1999) and a crossover operator based on selecting the 'best' events from each parent. Although the best memetic algorithm required a greater computation time than the tabu search, the solutions produced are nearly always of a higher quality. This approach is discussed further in section 4.

Dias et al. (Dias et al., 2003) developed a tabu search and a genetic algorithm for solving rostering problems in Brazilian hospitals. Tests showed the genetic algorithm slightly outperformed the tabu search but, in practice, both approaches were welcomed by the hospital users without preference as they were both significantly superior to manual efforts. A user interface which easily allowed small changes to the schedule by hand was also appreciated by the staff.

Although genetic and memetic algorithms have proven to be popular methodologies for personnel scheduling, a wide variety of other Operations Research and Artificial Intelligence methods have also been used to produce high quality rosters. These include mathematical programming (Thornton and Sattar, 1997; Bard and Purnomo, 2007) and in particular and more recently column generation and branch and price methods (Jaumard et al., 1998; Bard and Purnomo, 2005). Constraint programming (Meyer auf'm Hofe, 2000; Meisels and Schaerf, 2003), local search and other metaheuristics (Dowsland and Thompson, 2000; Bellanti et al., 2004; Burke et al., 2004; Burke et al., 2008) have also been successful. More novel approaches include Bayesian optimisation algorithms (Aickelin et al., 2007), case-based reasoning (Beddoe and Petrovic, 2006; Beddoe and Petrovic, 2007), hyperheuristics (Burke et al., 2003) and Pareto optimisation (Jaszkiewicz, 1997). Overviews of many of these approaches and others can be found in a range of survey papers (Burke et al., 2004; Ernst et al., 2004; Ernst et al., 2004).

## 2 The Nurse Rostering Problem and Benchmark Instances

Simply stated, the nurse rostering problem requires the assignment of shifts to personnel to ensure that sufficient employees are present to perform the duties required. There are a number of constraints such as working regulations and legal requirements and a number of objectives such as maximising the nurses working preferences and/or minimising wage costs.

The instances we are solving are benchmark data sets taken from a number of real world scenarios. An xml data format has been developed for the purpose of presenting and sharing

complex personnel scheduling instances and solutions. All the instances along with best known solutions and a complete description of the problem are publicly available at the research website (http://www.cs.nott.ac.uk/~tec/NRP). Source code for the objective functions, parsers and software for viewing and manually solving the test instances is also provided. A screenshot of this software is given in Figure 1. Making this software freely available for other researchers encourages investigation into a highly practical and scientifically challenging problem. It also helps ensure the accuracy of results and aids the verification of new solutions.

Figure 1 About here

The instances used in the experiments are listed in Table 1.

Table 1. About here

The instances provide a diverse and challenging collection. They vary not only in size (in terms of number of nurses and shift types) but also in the type, number and priority of constraints and objectives in each instance. The 'BCV' instances are taken from hospitals in Belgium. The other instances are from researchers and industrial collaborators across six different countries.

Some of the smaller instances e.g. BCV-5.4.1, LLR and GPost-B can be solved to optimality using CPLEX 10 (although the formulation is important for each instance). For some of the larger instances, the best known solutions have been proven to be optimal. This was done by producing lower bounds using various (often instance specific) relaxations and decompositions which were again solved using CPLEX 10. Many of the best known solutions were found using an ejection chain based method (Burke et al., 2007) (often over a very long execution time). This method is also used as the improvement method in the scatter search presented here (see section 3.2).

For all the instances, the cover requirements (nurse demand) are specified per shift type for each day in the planning period. Over cover and under cover is not permitted. Another hard constraint is that a nurse may not work more than one shift of the same type on the same day (this ensures solution structure). For most of the instances, employees may also only work one

shift per day. The only other hard constraint is that shifts which require skills are assigned to a nurses with the required skills.

Every other constraint is soft, that is, it is part of the objective function. This means it may be violated but a value will be added in the objective function which is proportional to the severity (size) of the violation and the relative importance of the constraint (set using weights). The objective function is a minimisation of a weighted sum of the soft constraints.
Each employee can have a unique set of constraints (and priorities) for their schedule (work pattern). This considerably increases the flexibility of the model but also the complexity. Examples of constraints from the BVC instances alone 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 (a weekend definition is also a user definable parameter, that is, 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, that is, 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, it may be desirable to avoid a late shift after an early shift.
- Maximum total number of assignments for all Mondays, Tuesdays, Wednesdays… For example, a nurse may request not to work on Sundays or may require to work a maximum of two Fridays 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.
- Day on/off or specific shift on/off requests with associated priorities.

The mathematical formulation of these objectives can be found in (Burke et al., 2008) (it is not repeated here due to its considerable size of 29 pages). However, to help further understanding of the problem, an IP model of the ORTEC01 instance is provided. This model is based on the one originally published in (Burke et al., 2008). To accurately describe our framework though, some of the constraints have been changed to objectives (goals). In our framework these goals are then given very high weights, such as 10000.

Parameters:

$I$ = Set of nurses available.

$I_t \mid t \in \{1,2,3\}$ = Subset of nurses that work 20, 32, 36 hours per week respectively, $I = I_1 + I_2 + I_3$.

$J$ = Set of indices of the last day of each week within the scheduling period = {7, 14, 21, 28, 35}.

$K$ = Set of shift types = {1(*early*), 2(*day*), 3(*late*), 4(*night*)}.

$K'$ = Set of undesirable shift type successions = {(2,1), (3,1), (3,2), (1,4), (4,1), (4,2), (4,3)}.

$d_{jk}$ = Coverage requirement of shift type $k$ on day $j$, $j \in \{1,\ldots,7|J|\}$.

$m_i$ = Maximum number of working days for nurse $i$.

$n_1$ = Maximum number of consecutive *night* shifts.

$n_2$ = Maximum number of consecutive working days.

$c_k$ = Desirable upper bound of consecutive assignments of shift type $k$.

$g_t$ = Desirable upper bound of weekly working days for the $t$-th subset of nurses.

$h_t$ = Desirable lower bound of weekly working days for the $t$-th subset of nurses.

Decision variables:

$x_{ijk}$ = 1 if nurse $i$ is assigned shift type $k$ for day $j$, 0 otherwise

The constraints are:

1. Shift cover requirements.

$$\sum_{i \in I} x_{ijk} = d_{jk}, \qquad \forall j \in \{1,...,7|J|\}, k \in K$$

2. A nurse may not start more than one shift each day.

$$\sum_{k \in K} x_{ijk} \leq 1, \quad \forall i \in I, j \in \{1,...,7|J|\}$$

The soft constraints are formulated as (weighted $w_i$) goals. The overall objective function is:

$$\text{Min } \overline{G}(x) = \sum_{i=1}^{\Delta} w_i \overline{g}_i(x),$$

Where the goals are:

1. Complete weekends (i.e. Saturday and Sunday are both working days or both off).

$$\overline{g}_1(x) = \left| \sum_{i \in I} \sum_{j \in J} \left| \sum_{k \in K} [x_{i(j-1)k} - x_{ijk}] \right| \right|$$

2. Minimum of two consecutive non-working days

$$\overline{g}_2(x) = \sum_{i \in I} \sum_{j=2}^{7|J|-1} \max\left\{0, \sum_{k \in K} [-x_{i(j-1)k} + x_{ijk} - x_{i(j+1)k}]\right\}$$

3. A minimum number of days off after a series of shifts.

$$\overline{g}_3(x) = \sum_{i \in I} \sum_{j=2}^{7|J|-1} \max\left\{0, \sum_{k \in K} [x_{i(j-1)k} - x_{ijk} + x_{i(j+1)k}] - 1\right\}$$

4. A maximum number of consecutive shifts of type early and late.

$$\overline{g}_4(x) = \sum_{i \in I} \sum_{r=1}^{7|J|-c_k} \sum_{k \in \{1,3\}} \max\left\{0, \sum_{j=r}^{r+c_k} x_{ijk} - c_k\right\}$$

5. A minimum number of consecutive shifts of type early and late.

$$\overline{g}_5(x) = \sum_{i \in I} \sum_{j=2}^{7|J|-1} \sum_{k \in \{1,3\}} \max\{0, -x_{i(j-1)k} + x_{ijk} - x_{i(j+1)k}\}$$

6. A maximum and minimum number of working days per week.

$$\overline{g}_6(x) = \sum_{t=1}^{3} \sum_{i \in I_t} \sum_{w=1}^{|J|} \left[ \max\left\{0, \sum_{j=7w-6}^{7w} \sum_{k \in K} x_{ijk} - g_t \right\} + \max\left\{0, h_t - \sum_{j=7w-6}^{7w} \sum_{k \in K} x_{ijk} \right\} \right]$$

7. A maximum of three consecutive working days for part time nurses.

$$\overline{g}_7(x) = \sum_{i \in I_1} \sum_{r=1}^{7|J|-3} \max\left\{0, \sum_{j=r}^{r+3} \sum_{k \in K} x_{ijk} - 3 \right\}$$

8. Avoiding certain shift successions (e.g. an *early* shift after a *day* shift).

$$\overline{g}_8(x) = \sum_{i \in I} \sum_{j=1}^{7|J|-1} \sum_{(k_1, k_2) \in K'} \max\left\{0, x_{ijk_1} + x_{i(j+1)k_2} - 2\right\}$$

9. Maximum number of working days.

$$\overline{g}_9(x) = \sum_{i \in I} \max\left\{0, \sum_{j=1}^{7|J|} \sum_{k \in K} x_{ijk} - m_i \right\}$$

10. Maximum of three working weekends.

$$\overline{g}_{10}(x) = \sum_{i \in I} \max\left\{0, \sum_{j \in J} \sum_{k \in K} \max\{x_{i(j-1)k}, x_{ijk}\} - 3\right\}$$

11. Maximum of three night shifts.

$$\overline{g}_{11}(x) = \sum_{i \in I} \max\left\{0, \sum_{j=1}^{7|J|} x_{ij4} - 3\right\}$$

12. A minimum of two consecutive night shifts.

$$\overline{g}_{12}(x) = \sum_{i \in I} \sum_{j=2}^{7|J|-1} \max\left\{0, -x_{i(j-1)4} + x_{ij4} - x_{i(j+1)4}\right\}$$

13. A minimum of two days off after a series of consecutive night shifts. This is equivalent to avoiding the pattern: night shift – day off – day on.

$$\overline{g}_{13}(x) = \sum_{i \in I} \sum_{j=2}^{7|J|-1} \max\left\{0, x_{i(j-1)4} - \sum_{k \in K} x_{ijk} + \sum_{k \in K} x_{i(j+1)k} - 1\right\}$$

14. Maximum number of consecutive night shifts.

$$\overline{g}_{14}(x) = \sum_{i \in I} \sum_{r=1}^{7|J|-n_1} \max\left\{0, \sum_{j=r}^{r+n_1} x_{ij4} - n_1\right\}$$

15. Maximum number of consecutive working days.

$$\overline{g}_{15}(x) = \sum_{i \in I} \sum_{r=1}^{7|J|-n_2} \max\left\{0, \sum_{j=r}^{r+n_2} \sum_{k \in K} x_{ijk} - n_2\right\}$$

# 3 The Scatter Search for Nurse Rostering Problems

In Glover's template for scatter search (Glover, 1998), five component subroutines for the overall process are outlined. The following sections describe an implementation of these subroutines for the nurse rostering problem. The overall scatter search is outlined in Figure 2.

Figure 2 about here

## 3.1 The Diversification Generation Method

A diversification generation method is required to create a diverse set of solutions. These solutions are then improved (according to the objective function) and added to the initial reference set subject to certain criteria. When creating the diverse set of solutions, the objective value for each solution is not examined, only its similarity to other solutions in the diverse set is of interest. The method used for creating a diverse solution set is outlined in Figure 3.

Figure 3. about here.

The method constructs new rosters based on the structure of other rosters in the set. At each iteration of the roster construction method, the nurse to assign a shift to is selected by examining who has been assigned that shift the *least* in the other rosters in the set.

As the only hard constraints are related to cover and the number of shifts that can be worked each day, this method will always return a feasible solution if one exists. Whether a feasible solution exists can be determined at the start of the algorithm by a simple calculation based on the cover requirements and the number of nurses available. If the instance is determined to be infeasible the user is notified and asked to either adjust the cover constraints and/or add more nurses to the roster.

To measure the similarity of two rosters, a simple but effective method is used: counting the number of *nurse to shift* assignments in common. An example of this is given in Figure 4 which shows the individual schedules of three nurses (labelled A, B, and C) from two different rosters. The lettered squares labelled E, D, L, N represent different shifts (Early, Day, Late and Night). Identical assignments are highlighted, for example nurse A has a day

shift (D) on Saturday 4th in both schedules. In this example, just looking at these three nurses' schedules, there are 23 identical assignments in the two rosters.

Figure 4 about here.

### 3.2 Improvement Method

The goal of the improvement method is to try to improve any solutions according to their objective function. The solutions which it works upon may be those produced by the diversification generation method (see section 3.1) or the solution combination method at the start of the algorithm (see section 3.5). Two different algorithms were tested as the improvement method. The first one is a hill climbing algorithm which has a very short execution time (< 1 second even on the largest instances) but on its own is unable to produce satisfactory solutions. It uses a single shift move neighbourhood commonly used in nurse rostering approaches e.g. (Jaszkiewicz, 1997; Meisels and Schaerf, 2003). An example of this move is given in Figure 5 where a day shift (D) on the 3rd is moved from employee D to employee A. The pseudocode for the hill climber is provided in Figure 6.

Figure 5 about here.
Figure 6 about here.

The second improvement method tested is the time predefined variable depth search presented in (Burke et al., 2007). The variable depth search belongs to the family of ejection chain methods. It works by chaining together single swaps of shifts between nurses. Chaining swaps together (that is, performing swaps concurrently) allows the search to escape from local optima that the hill climber gets stuck in. Hence it is a more powerful method. The variable depth search uses a number of heuristics to select the swaps to chain together and to decide whether to continue a current chain. The algorithm also accepts a predefined maximum run time. This makes it easily adaptable as the improvement method for our scatter search.

### 3.3 Reference Set Update Method

The reference set update method is used at two separate stages in the algorithm. It is used to create the initial reference set from the solutions produced by the diversification method (the

diverse set of solutions). Afterwards, it is used to maintain the reference set between generations. It decides whether to add to the reference set new solutions that are produced by the combination and improvement methods.

The reference set is initialised in a similar manner to that used by Glover et al. (Glover et al., 2003). After all the solutions in the diverse set of solutions are improved by the improvement method, they are ranked according to the objective function. The best $b_1$ of these solutions are then added to the reference set. From the remaining solutions, $b_2$ are selected and added, based on their contribution to the diversity of the reference set. Figure 7 outlines the process.

When the update method is used between generations (that is, after the solution combination method), new solutions are added to the reference set if their objective function value is better than the reference set's current worst solution and the set does not already contain an identical solution. If a new solution is added, the current worst solution is removed. The contribution to the set's diversity is no longer considered. So, in effect, the $b_2$ tier is merged into the $b_1$ tier. This design decision was made after preliminary testing in which it was observed that including the $b_2$ tier produced much longer run times. In order to develop a practical approach it was necessary to keep the algorithm's running time to preferably less than 15 minutes.

Figure 7 about here.

### 3.4 Subset Generation Method

The subset generation method is used to identify the subsets of solutions in the reference set that will be used by the combination method to create new solutions. A commonly used subset generation method in scatter search is that suggested by Glover (Glover, 1998). This approach is also adopted here. Using this method, four different types of subsets of increasing size are identified. They are:

1. All unique subsets of the reference set containing 2 elements.
2. Subsets of size 3 identified by adding to each 2-element subset (above) the best solution not already in this subset.
3. Subsets of size 4 identified by adding to each 3-element subset (above) the best solution not already in this subset.
4. Subsets containing the best *i* solutions, for i = 5 to |RefSet|

The best solutions here refer only to the objective function values. At each iteration, it is also necessary to keep a record of which solutions in the reference set are new. This avoids combining sets of old solutions which were already combined in the previous iteration.

### 3.5 The Solution Combination Method

The solution combination method uses two or more solutions (selected by the subset generation method) for reference and produces one or more new solutions, often using a path relinking mechanism. These new solutions are then improved by the improvement method and then either added to the reference set or discarded.

Although the subset generation method can be easily adapted to a wide range of problems, the solution combination method is often more specifically designed for each problem. Glover et al. (Glover et al., 2000) discuss a number of forms that the solution combinations or path relinking could take. The solution combination method developed here is categorized in their paper as a constructive neighbourhood approach "*where the guiding solutions vote for attributes to be included in the initiating solution*" (Glover et al., 2000). In our case, the attributes are *shift to nurse* allocations within the rosters.

A solution can be regarded as simply a number of *shift to nurse* assignments. In the solution combination method, each *shift to nurse* allocation in each solution to be combined is regarded as a *vote* for a *candidate*. The candidates available for selection are all the possible *shift to nurse* assignments for the problem instance. All these votes are then analysed and used to construct a new solution. The shift assignments (candidates) for the new solution are made according to the number of votes they received from the guiding solutions. The pseudocode in Figure 8 outlines the process.

In Figure 8, a candidate is a *shift to nurse* assignment on a specific day. The voters are the guiding solutions, each solution is a voter and each assignment within that solution is a vote for a specific candidate. The first step in the process is to create a new solution which initially has no assignments.

At step 5 of Figure 8, the list of candidates (that is, *shift to nurse* assignments) is sorted by the number of votes they received. As the guiding sets of solutions are small (see the subset definitions), the candidates are often tied. If this is the case, two tie-breakers are used. If two candidates receive the same number of votes, the candidate whose voters (guiding solutions) have had the least total number of successful votes (assignments) is ordered first. If this does

not differentiate between the two candidates, then the candidate whose voters have the lowest sum of objective function values comes first (lowest as it is a minimisation problem).
The requirement at step 7 ensures that cover will not be exceeded. However, it is possible that the new solution may be infeasible through cover being under-satisfied. If this happens it is repaired by a simple greedy assignment algorithm before the improvement method is applied (that is, the hill climber or variable depth search).

Figure 8 about here.
Figure 9 about here.

As an example of this scheme and how the votes are ranked, Figure 9 shows a set of six small example rosters. The rosters contain day (D) and night (N) shifts assigned to four employees (labelled A-D) over a five day period (Monday-Friday). Table 2 lists all the *shift to nurse* assignments in these six rosters and ranks them by the number of times they appear in the set of rosters. For example, employee A is assigned a day shift on Monday in five of the six rosters. Employee B is assigned a night shift on Monday in only one of the rosters and so on. This table is used to construct the new roster. The first assignment made in the new roster will be a day shift for nurse A on day one. The second assignment will be a day shift for nurse A on day two etc. When the count is the same, the tie-breakers listed at step 5 in Figure 8 are used.

Table 2 about here.

# 4 Results

The algorithm is tested using the benchmark data sets introduced in section 2. The first experiments investigate the benefit and efficacy of the diversification method, the improvement methods and the combination method. We then compare the scatter search against three previously published nurse rostering algorithms.
The following parameters were used for the scatter search: a reference set of size five ($b_1$=3, $b_2$=2, initial number of solutions=8). When the variable depth search was used as the improvement method it was assigned a maximum execution time of five seconds. These parameters were chosen after preliminary testing in order to produce an algorithm with a

practical execution time (preferably less than 15 minutes). For example, for some of the instances using a reference set of size ten or a variable depth search of five minutes produced a run time of over 24 hours. In practice, end users strongly resist these very long execution times. The parameters chosen also produced a run time closer to that of the other approaches so enabling more informative comparisons. The experiments were performed on a desktop PC with an Intel Core 2 Duo 2.83GHz processor.

The first experiment conducted was designed to investigate the effect of the improvement method. The scatter search was tested using the hill climber as the improvement method (SS1) and then with the variable depth search as the improvement method (SS2). The best, average and average time (in seconds) for five tests on each instance were recorded. The results are shown in the column "Standard scatter search" of Table 3. Optimal solutions are in bold, best known solutions are in italics. As can be seen, SS2 has a longer execution time on all instances but the increase in computation time does result in better solutions. Comparing best results, SS2 is equal to SS1 on five instances (in which both algorithms find optimal solutions) and better on all the other instances. SS2 is also able to find optimal solutions for a further three instances and equal the best known on another two. The average results for SS2 are also better than SS1 on all instances and better than or equal to all but one of SS1's best results of the five trials. Although SS1 can produce optimal solutions on some instances, if the end user can afford the extra computation time SS2 is a better algorithm.

The second experiment was designed to investigate the benefit of the diversification method used at the start of the algorithm. Again SS1 and SS2 were both tested on all instances but this time the diversification method was removed and the initial diverse set of solutions was generated using a random shift assignment method. The random shift assignment method works by repeatedly assigning randomly selected shifts to randomly selected employees until cover is satisfied. The results of these tests are shown in Table 3 in the "No diversification method" column. Examining the results it is evident that for these tests the absence of the diversification method has no effect on the quality of solutions found. Comparing 'with the diversification method' versus 'without the diversification method', examining the 'Best' column, they outperform each other the same number of times (for SS1 and SS2). The average results on each instance are also very similar.

The third experiment investigated the benefit of the solution combination method presented in section 3.5. Again, SS1 and SS2 were tested on all instances but this time the solution combination method (used to create a new solution from parent solutions) was removed. In its place, new solutions were created using a random shift assignment method (that is, randomly assign shifts until cover is satisfied). As before, these solutions are then improved using the improvement method before being added (or not) to the reference set. The results of these tests are shown in the "No solution combination" column of Table 3. For SS1, on all instances not using the solution combination method produces a shorter run time. However, on all instances the solutions are also worse (for both average and best results). The shorter run time is due to a lower number of generations which in turn is due to the poor quality solutions created by the random assignment combination method. As these solutions are poor quality, after only a few generations, there are none of sufficient quality to enter the reference set after the combination method plus improvement method phase. This shows the combination method developed is effective. However, for SS2, not using the combination method has little effect on the solution quality and run time. This is because the more powerful improvement method in SS2 is able to improve the poor quality solutions generated by the random assignment combination method to a similar level as when the 'voting' combination method is used.

Table 3 about here.

To provide a comparison of the scatter search against other algorithms, Table 4 contains the best published results of a constructive approach of Brucker et al. (Brucker et al., 2009) and the best published result of a hybrid variable neighbourhood search of Burke et al. (Burke et al., 2008). These are currently the only published results on the benchmark instances. To provide an additional comparison we have implemented the memetic algorithm of Burke et al. (Burke et al., 2001) and included the best, average and average time (in seconds) of five tests on each instance.

The memetic algorithm (MEH) (Burke et al., 2001) is a hybrid approach which performs a tabu search on individuals in the population between generations and a greedy shuffling step on the best solution at the end. It was originally developed as part of the software package from which the BCV instances were obtained. It was shown to be a robust approach and the

best method on more difficult instances. The same settings as described in the original paper were used (underlying memetic algorithm ME4, population size of twelve and stop criterion of no improvement during two generations).

The constructive approach (Brucker et al., 2009) is an iterative process in which a small number of shifts are heuristically assigned before a local search similar to the hill climber used here is applied. After the local search, more shifts are assigned and then the local search reapplied and so on. The variable neighbourhood search (VNS) (Burke et al., 2008) is also an iterative hybrid approach. A variable neighbourhood descent search is followed by a phase of heuristically selecting and reassigning some of the worse employee schedules. It was shown to outperform the genetic algorithm that formed part of a commercial rostering package. The memetic algorithm and the scatter search experiments were performed on a desktop PC with an Intel Core 2 Duo 2.83GHz processor. The constructive approach and VNS algorithms were executed on a desktop PC with an Intel Pentium 4 2.4GHz processor.

Table 4 about here.

Comparing the memetic algorithm (MEH) against the scatter search using the hill climber (SS1), MEH produces better solutions on all instances when looking at the 'average' column and better or equal solutions on all instances when looking at the 'best' column. For the five instances for which both algorithms produced equal quality solutions, the solutions were optimal. However, MEH does use more computation time on all instances. Comparing MEH with the scatter search using the variable depth search (SS2), for the 'best' column, SS2 outperforms on all instances apart from the six on which both algorithms find optimal solutions. Looking at the 'average' column, SS2 is better on all but five on which both algorithms produce optimal solutions. Examining computation times, SS2 uses more time on thirteen of the twenty instances and MEH takes longer on six.

Comparing the scatter search against the constructive approach of Brucker et al. (Brucker et al., 2009), SS1 is better or equal on all instances in less computation time (using the best of five trials and where total time = average time * 5). However, the constructive approach experiments were performed on a slower computer. Compared to the hybrid VNS of Burke et al. (Burke et al., 2008) on the instance ORTEC01, SS1 is a lot worse but has a significantly lower computation time. SS2 outperforms the hybrid VNS on the instance ORTEC01 in a lower computation time. Again, the hybrid VNS experiments were performed on a slower computer but in this case it is clear the small difference in computer power cannot alone account for the much better result of SS2 (twelve hours versus approximately one hour for all five trials).

# 5 Conclusions

A scatter search has been presented for the nurse rostering problem and evaluated using benchmark instances. Applying the approach to the nurse rostering problem required the development of new methods for measuring similarity between solutions, creating solutions from multiple parents and creating diversity in the initial population. For measuring similarity, a simple but effective method was used : counting the number of common shift assignments in rosters. To create new solutions for each generation a 'democratic' approach was adopted in which each guiding solution 'votes' for the assignments in the new solutions. Experiments showed that this combination method was an effective heuristic when the scatter search had a less powerful improvement method. However, the experiments also indicated

that the diversification method used at the start of the algorithm had little effect on the final solution quality. When the scatter search was used with the more powerful improvement method (a variable depth search) the solutions were of higher quality but the computation time increased. The scatter search with a less powerful improvement method (a hill climber) was still able to produce optimal solutions for a number of the real world benchmark instances. Even with the more powerful improvement method the scatter search was able to find very high quality solutions (eight optimal and two more equal to the best known) in practical computation times.

When compared with Burke et al.'s memetic algorithm, the scatter search with hill climber improvement method (SS1) underperforms (but had a shorter computation time). When comparing the memetic algorithm with the scatter search with the variable depth search as the improvement method (SS2), the scatter search outperforms or is equal on all instances but has a longer computation time on twelve of the twenty instances. Compared to the heuristic constructive method of Brucker et al., SS1 is better or equal in less time on all but one instance but on a faster computer. Compared to the hybrid approach of Burke et al. on instance ORTEC01, SS1 is worse in less time and SS2 is better but also in significantly less time but again on a faster computer. These results against previously published algorithms are a good indication that the scatter search is an efficient and successful approach. The solution similarity comparison method is simple and intuitive and the solution combination method is also easily understandable. When these subroutines are combined into the overall scatter search, a relatively straightforward yet demonstrably effective approach is produced.

All the test instances and best solutions are publicly available at the research website (http://www.cs.nott.ac.uk/~tec/NRP/).

**Acknowledgements**

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

**References**

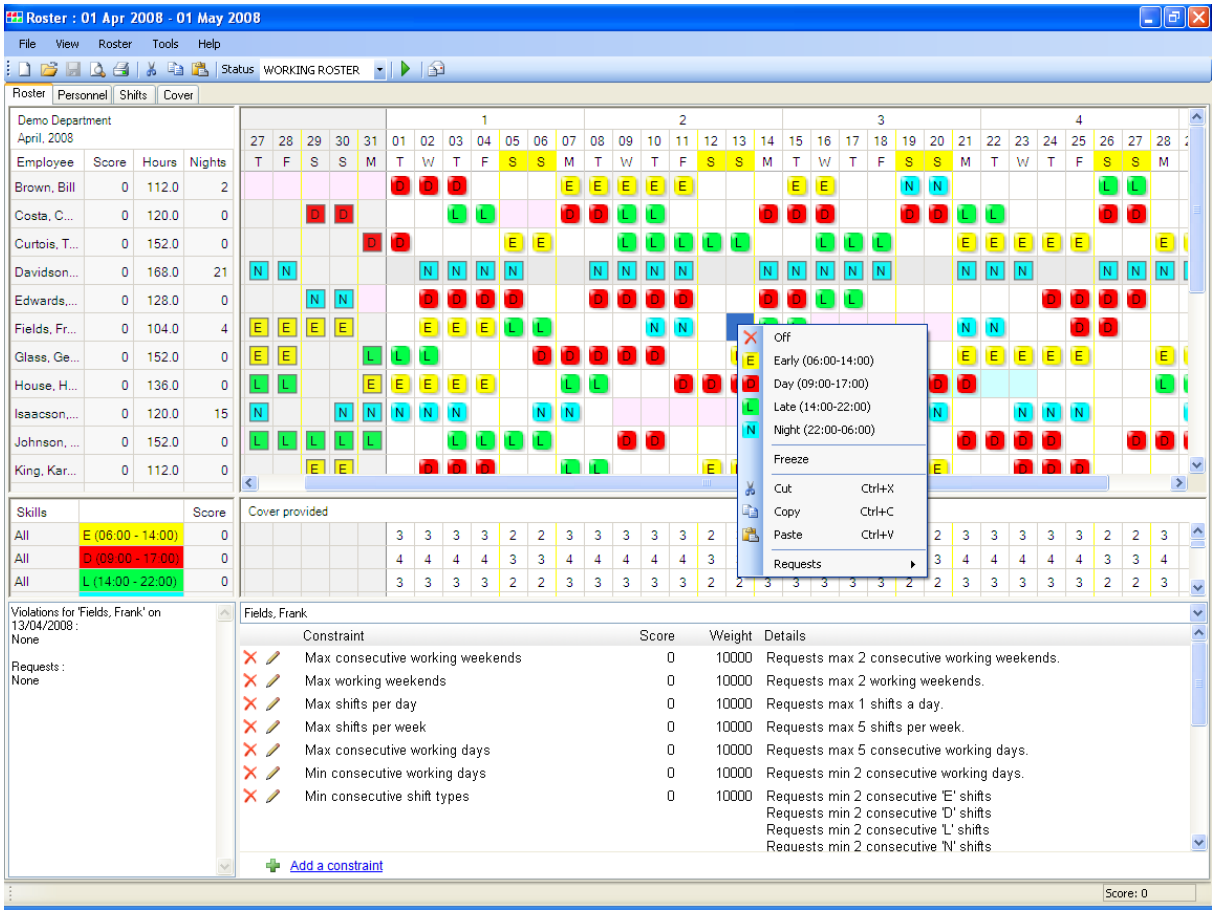


Figure 1 Rostering software screenshot

1. Create initial set of diverse solutions
2. Improve each solution in diverse set
3. Create initial reference set (*RefSet1*) from the diverse solutions
4. Make a copy of the reference set (*RefSet2*)
5. FOR each untried subset of solutions in *RefSet1*
6. Combine solutions in subset to produce a new solution (*NewSolution*)
7. Improve *NewSolution*
8. Replace a solution in *RefSet2* with *NewSolution* subject to certain criteria
9. ENDFOR
10. IF *RefSet1* and *RefSet2* are not identical
11. SET *RefSet1* := *RefSet2*
12. GOTO 4.
13. ENDIF
14. Return the best solution in *RefSet* or GOTO 1.

Figure 2 Scatter search overview

1. Create an empty set (*set*) of size n for the diverse solutions
2. UNTIL *set* is full
3. Create a roster (*roster*) with no assignments made
4. FOR each day (*day*) in *roster*
5. FOR each shift type (*shift*) to be covered on *day*
6. UNTIL the cover is satisfied
7. Assign *shift* to a nurse who has been assigned *shift* on *day* the least number of times in all other rosters in *set*
8. If more than one nurse has received *shift* on *day* the least number of times then randomly select one of them
9. ENDUNTIL
10. ENDFOR
11. ENDFOR
12. Add *roster* to *set*
13. ENDUNTIL

Figure 3 Pseudocode for the scatter search initial set creation

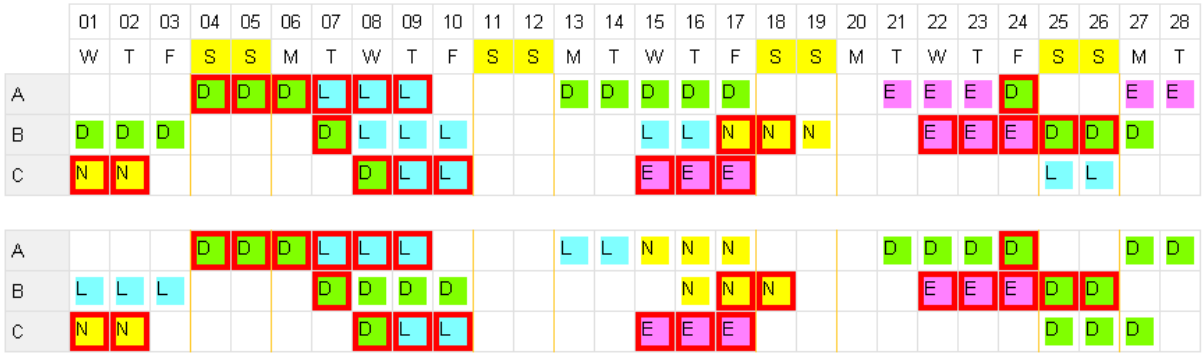


Figure 4 Example of the roster similarity measure

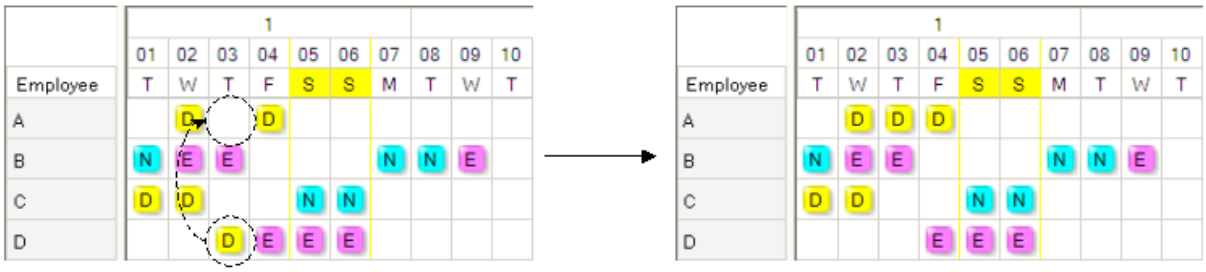


Figure 5 Example of the neighbourhood operator for the hill climbing algorithm

```
1.  WHILE there are untried swaps
2.     FOR each employee (E1) in the roster
3.       FOR each day (D1) in the planning period
4.         FOR each employee (E2) in the roster
5.           Swap assignments between E1 and E2 on D1
6.           IF roster objective function decreases THEN
7.             Break from this loop and move on to the next day
8.           ELSE
9.             Reverse the swap
10.          ENDIF
11.        ENDFOR
12.      ENDFOR
13.    ENDFOR
14. ENDWHILE
```

Figure 6 Hill climber pseudocode

*P* is the set of solutions created using the diversification generation method.
*RefSet* is the reference set and is initially empty.

$b_1$ and $b_2$ are algorithm specific parameters (integers >= 0).

1. FOR 1 to $b_1$
2. Select from *P* the best solution according to the objective function
3. Remove the solution from *P* and add it to *RefSet*
4. ENDFOR
5. FOR 1 to $b_2$
6. For each solution in *P* calculate its total similarity to all the solutions currently in *RefSet* (using the similarity measure defined in section 3.1)
7. Select the least similar solution (the schedule with least assignments in common with other rosters in *RefSet*)
8. Remove the solution from *P* and add to *RefSet*
9. ENDFOR

Figure 7 Scatter search reference set initialisation

1. Identify *candidates* as the set of all possible shift to nurse assignments for this instance
2. Collect all the candidates' votes from each solution in the guiding set
3. Remove from *candidates* any candidate with zero votes
4. IF *candidates* is empty
   GOTO 10.
5. Sort *candidates* by:
   a) decreasing total number of votes
   b) increasing total number of votes successful for voters selecting this candidate
   c) increasing sum of objective function values for voters selecting this candidate
6. Select and remove the first candidate in *candidates*
7. Make the assignment represented by this candidate in the new solution unless it exceeds cover requirements
8. IF the assignment was made
   GOTO 4.
9. IF *candidates* is not empty
   GOTO 6.
10. Return new solution

Figure 8 An outline of the scatter search solution combination method


Figure 9 Example set of rosters

| Instance | No. nurses | No. shift types | Planning period (days) | Best known solution |
|---|---|---|---|---|
| ORTEC01 | 16 | 4 | 31 | **270** |
| BCV-1.8.1 | 8 | 4 | 28 | 252 |
| BCV-2.46.1 | 46 | 4 | 28 | **1572** |
| BCV-3.46.1 | 46 | 3 | 26 | 3280 |
| BCV-3.46.2 | 46 | 3 | 26 | 894 |
| BCV-4.13.1 | 13 | 4 | 29 | **10** |
| BCV-5.4.1 | 4 | 4 | 28 | **48** |
| BCV-6.13.1 | 13 | 4 | 30 | 768 |
| BCV-7.10.1 | 10 | 6 | 28 | **381** |
| BCV-8.13.1 | 13 | 4 | 28 | **148** |

| | | | | |
|---|---|---|---|---|
| BCV-A.12.1 | 12 | 4 | 31 | 1294 |
| BCV-A.12.2 | 12 | 4 | 31 | 1953 |
| GPost | 8 | 2 | 28 | **5** |
| GPost-B | 8 | 2 | 28 | **3** |
| LLR | 27 | 3 | 7 | **301** |
| Millar-2Shift-DATA1 | 8 | 2 | 14 | **0** |
| Millar-2Shift-DATA1.1 | 8 | 2 | 14 | **0** |
| QMC-1 | 19 | 3 | 28 | 16 |
| SINTEF | 24 | 5 | 21 | **0** |
| Valouxis-1 | 16 | 4 | 31 | **20** |

Table 1 Test Instances (values in **bold** are proven optimal)

| Assignment (Nurse-Day-Shift) | Count | Assignment (Nurse-Day-Shift) | Count |
|---|---|---|---|
| A-1-D | 5 | A-3-D | 2 |
| A-2-D | 5 | A-4-D | 2 |
| B-5-D | 5 | D-1-N | 2 |
| C-5-D | 5 | D-2-N | 2 |
| A-5-D | 4 | D-3-N | 2 |
| B-3-D | 4 | D-4-N | 2 |
| B-4-D | 4 | D-5-N | 2 |
| D-5-D | 4 | B-1-N | 1 |
| C-1-N | 3 | B-2-N | 1 |
| C-2-N | 3 | B-3-N | 1 |
| C-3-D | 3 | B-4-N | 1 |
| C-4-D | 3 | B-5-N | 1 |
| D-3-D | 3 | C-3-N | 1 |
| D-4-D | 3 | C-4-N | 1 |
| A-3-N | 2 | C-5-N | 1 |
| A-4-N | 2 | D-1-D | 1 |
| A-5-N | 2 | D-2-D | 1 |

Table 2 Ranking assignments

| Algorithm | Instance | Standard scatter search | | | No diversification method | | | No solution combination | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Avg. | Avg. time | Best | Avg. | Avg. time | Best | Avg. | Avg. time |
| SS1 | ORTEC01 | 3377 | 4248 | 142 | 3101 | 5460 | 129 | 9848 | 10596 | 36 |
| | BCV-1.8.1 | 272 | 288 | 19 | 277 | 289 | 20 | 327 | 355 | 6 |
| | BCV-2.46.1 | ***1572*** | 1587 | 296 | 1585 | 1595 | 341 | 1593 | 1604 | 140 |
| | BCV-3.46.1 | 3565 | 3631 | 482 | 3625 | 3677 | 512 | 3945 | 3969 | 150 |
| | BCV-3.46.2 | 908 | 911 | 411 | 904 | 908 | 395 | 931 | 936 | 140 |
| | BCV-4.13.1 | 12 | 45 | 25 | ***10*** | 31 | 33 | 44 | 78 | 8 |
| | BCV-5.4.1 | ***48*** | 136 | 3 | 49 | 138 | 3 | 199 | 395 | 1 |
| | BCV-6.13.1 | 964 | 1060 | 52 | 958 | 1079 | 52 | 1423 | 1473 | 18 |
| | BCV-7.10.1 | ***381*** | 387 | 22 | 382 | 388 | 21 | 395 | 412 | 9 |
| | BCV-8.13.1 | ***148*** | 149 | 31 | ***148*** | 150 | 37 | 189 | 326 | 11 |
| | BCV-A.12.1 | 1880 | 2239 | 111 | 1979 | 2112 | 115 | 3800 | 4000 | 59 |
| | BCV-A.12.2 | 2528 | 2812 | 114 | 2750 | 2811 | 101 | 4007 | 4656 | 55 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | GPost | 2243 | 4901 | 10 | 3018 | 5547 | 10 | 9817 | 12041 | 3 |
| | GPost-B | 2693 | 3807 | 12 | 3127 | 4297 | 9 | 5969 | 8325 | 2 |
| | LLR | 317 | 329 | 12 | 308 | 322 | 10 | 373 | 378 | 3 |
| | Millar-2Shift-DATA1 | 400 | 580 | 2 | 200 | 460 | 2 | 1100 | 1160 | 1 |
| | Millar-2Shift-DATA1.1 | **0** | 100 | 2 | **0** | 80 | 2 | 200 | 280 | 1 |
| | QMC-1 | 50 | 54 | 56 | 50 | 56 | 56 | 86 | 96 | 17 |
| | SINTEF | 10 | 11 | 43 | 9 | 14 | 44 | 29 | 31 | 13 |
| | Valouxis-1 | 1620 | 2076 | 65 | 640 | 1976 | 70 | 1488 0 | 17608 | 14 |
| | | Standard scatter search | | | No diversification method | | | No solution combination | | |
| | | Best | Avg. | Avg. time | Best | Avg. | Avg. time | Best | Avg. | Avg. time |
| SS2 | ORTEC01 | 365 | 376 | 680 | 350 | 369 | 676 | 330 | 368 | 824 |
| | BCV-1.8.1 | *252* | 253 | 431 | *252* | *252* | 627 | *252* | *252* | 675 |
| | BCV-2.46.1 | ***1572*** | ***1572*** | 707 | ***1572*** | ***1572*** | 801 | ***1572*** | ***1572*** | 774 |
| | BCV-3.46.1 | 3351 | 3357 | 869 | 3334 | 3363 | 758 | 3336 | 3358 | 899 |
| | BCV-3.46.2 | *894* | *894* | 721 | *894* | *894* | 581 | *894* | *894* | 651 |
| | BCV-4.13.1 | ***10*** | ***10*** | 355 | ***10*** | ***10*** | 401 | ***10*** | ***10*** | 413 |
| | BCV-5.4.1 | ***48*** | ***48*** | 250 | ***48*** | ***48*** | 250 | ***48*** | ***48*** | 250 |
| | BCV-6.13.1 | 784 | 784 | 547 | 784 | 784 | 547 | 784 | 784 | 583 |
| | BCV-7.10.1 | ***381*** | 382 | 549 | ***381*** | ***381*** | 521 | ***381*** | ***381*** | 551 |
| | BCV-8.13.1 | ***148*** | ***148*** | 271 | ***148*** | ***148*** | 271 | ***148*** | ***148*** | 271 |
| | BCV-A.12.1 | 1600 | 1733 | 830 | 1645 | 1768 | 830 | 1593 | 1689 | 922 |
| | BCV-A.12.2 | 2180 | 2321 | 806 | 2278 | 2331 | 910 | 2255 | 2345 | 777 |
| | GPost | 9 | 9 | 861 | 8 | 9 | 755 | 9 | 9 | 821 |
| | GPost-B | 5 | 6 | 791 | 5 | 6 | 714 | 5 | 5 | 965 |
| | LLR | ***301*** | ***301*** | 423 | ***301*** | ***301*** | 447 | ***301*** | ***301*** | 459 |
| | Millar-2Shift-DATA1 | ***0*** | ***0*** | 182 | ***0*** | ***0*** | 267 | ***0*** | ***0*** | 183 |
| | Millar-2Shift-DATA1.1 | ***0*** | ***0*** | 4 | ***0*** | ***0*** | 19 | ***0*** | ***0*** | 15 |
| | QMC-1 | 20 | 22 | 887 | 31 | 1632 | 1048 | 29 | 1233 | 879 |
| | SINTEF | 4 | 5 | 821 | 4 | 5 | 718 | 4 | 6 | 781 |
| | Valouxis-1 | 100 | 104 | 800 | 100 | 116 | 744 | 80 | 112 | 730 |

Table 3 Scatter search results (optimal solutions are in **bold** best known are in *italics* )

| Instance | MEH (Burke et al., 2001) | | | Scatter search using hill climber (SS1) | | | Scatter search using variable depth search (SS2) | | | Constructive approach (Brucker et al., 2009) | | Hybrid VNS (Burke et al., 2008) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Average | Time (s) | Best | Average | Time (s) | Best | Average | Time (s) | Best | Time (s) | Best | Time |
| ORTEC01 | 535 | 1043 | 1516 | 3377 | 4248 | 142 | 365 | 376 | 680 | - | - | 541 | 12 hours |
| BCV-1.8.1 | 256 | 261 | 159 | 272 | 288 | 19 | *252* | 253 | 431 | 323 | 136 | - | - |
| BCV-2.46.1 | ***157*** ***2*** 336 | ***1572*** | 1787 | ***1572*** | 1587 | 296 | ***1572*** | ***1572*** | 707 | 1594 | 3424 | - | - |
| BCV-3.46.1 | 4 | 3387 | 8400 | 3565 | 3631 | 482 | 3351 | 3357 | 869 | 3601 | 2888 | - | - |
| BCV-3.46.2 | 900 | 902 | 2046 | 908 | 911 | 411 | *894* | *894* | 721 | - | - | | |
| BCV-4.13.1 | ***10*** | ***10*** | 112 | 12 | 45 | 25 | ***10*** | ***10*** | 355 | 18 | 208 | - | - |
| BCV-5.4.1 | ***48*** | ***48*** | 9 | ***48*** | 136 | 3 | ***48*** | ***48*** | 250 | 200 | 16 | - | - |
| BCV-6.13.1 | 875 | 930 | 370 | 964 | 1060 | 52 | 784 | 784 | 547 | 890 | 304 | - | - |
| BCV-7.10.1 | ***381*** | ***381*** | 83 | ***381*** | 387 | 22 | ***381*** | 382 | 549 | 396 | 216 | - | - |
| BCV-8.13.1 | ***148*** | ***148*** | 156 | ***148*** | 149 | 31 | ***148*** | ***148*** | 271 | ***148*** | 224 | - | - |
| BCV-A.12.1 | 164 0 | 1843 | 2740 | 1880 | 2239 | 111 | 1600 | 1733 | 830 | 3335 | 944 | - | - |
| BCV-A.12.2 | 246 5 | 2562 | 2989 | 2528 | 2812 | 114 | 2180 | 2321 | 806 | - | - | - | - |
| GPost | 915 | 1801 | 121 | 2243 | 4901 | 10 | 9 | 9 | 861 | - | - | - | - |
| GPost-B | 789 | 1826 | 95 | 2693 | 3807 | 12 | 5 | 6 | 791 | - | - | - | - |
| LLR | 305 | 306 | 38 | 317 | 329 | 12 | ***301*** | ***301*** | 423 | - | - | - | - |
| Millar-2Shift-DATA1 | 100 | 200 | 8 | 400 | 580 | 2 | ***0*** | ***0*** | 182 | - | - | - | - |
| Millar-2Shift-DATA1.1 | **0** | **0** | 4 | **0** | 100 | 2 | ***0*** | ***0*** | 4 | - | - | - | - |
| QMC-1 | 39 | 42 | 632 | 50 | 54 | 56 | 20 | 22 | 887 | - | - | - | - |
| SINTEF | 8 | 9 | 175 | 10 | 11 | 43 | 4 | 5 | 821 | - | - | - | - |
| Valouxis-1 | 560 | 832 | 593 | 1620 | 2076 | 65 | 100 | 104 | 800 | - | - | - | - |

Table 4 Benchmark results for scatter search and other approaches (optimal solutions are in **bold** best known are in *italics* )