A HyFlex Module for the Personnel Scheduling Problem

Tim Curtois, Gabriela Ochoa, Matthew Hyde, José Antonio Vázquez-Rodríguez
Automated Scheduling, Optimisation and Planning (ASAP) Group,
School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road,
Nottingham. NG8 1BB. UK

1. Problem Formulation

The personnel scheduling problem basically involves deciding at which times and on which days (i.e. which shifts) each employee should work over a specific planning period. However, the personnel scheduling problem is actually a title for a group of very similar problems. There is no general personnel scheduling problem. Instead there is a group of problems with a common structure but which differ in their constraints and objectives. This creates an additional challenge in implementing a problem domain module for personnel scheduling. To overcome this we have designed a data file format for which each instance can select a combination of a objectives and constraints from a wide choice. We then implemented a software framework containing all the functions for these constraints and objectives. The framework also contains methods for parsing these data files, data structures which can be used by heuristic algorithms (such as neighbourhood searches) and libraries for visualisations of instances and solutions.
As mentioned, there is a large and diverse collection of constraints and objectives that can appear in personnel scheduling problems. For example, in one problem there may be a constraint on the maximum number of hours a nurse can work in the planning period. In another problem though this constraint may be an objective. That is, the nurse is allowed to exceed a certain number of hours but the excess should be minimised. The objective may then also be given a priority (relative to other objectives) using a weight. To be able to handle all these variations yet at the same time minimise the complexity of the file format and the amount of programming necessary, a key design decision was made: *All constraints are modelled as weighted objectives*. When modelling a problem which contains constraints, the constraints are transformed to objectives with very high weights. As the weight is very high it is simple to tell if a solution is feasible or not just by examining the objective function value. The overall objective function is a weighted sum of all the sub-objectives.

The objectives for nurse rostering problems can be categorised into two groups:

Coverage objectives. These objectives aim to ensure that the preferred number of employees (possibly with skills) are working during each shift. Minimum and maximum levels of cover can also be set.

Employee working objectives. This group of objectives relates to the individual work patterns (schedules) for each employee. They aim to maximise the employees' satisfaction with their work schedules. Example objectives within this group include:

- Minimum/maximum number of hours worked.
- Minimum/maximum number of days on or off.
- Minimum/maximum number of consecutive working days.
- Minimum/maximum number of consecutive days off.
- Minimum/maximum number of consecutive working weekends.
- Minimum/maximum number of consecutive weekends off.
- Minimum/maximum number of shifts of a certain type. For example *night* shifts.
- Minimum/maximum number of consecutive shifts of a certain type. For example consecutive *night* shifts.
- Shift rotations. For example *early* shifts after *late* shifts should be avoided.
- Satisfying requests for specific shifts/days on or off.

To provide a more formal explanation of the problem we have included an IP model for the ORTEC01 instance. This problem was originally examined in [4] and subsequently in [8]. As according to our format, the constraints in the original formulation have been changed to objectives. The only constraint is that each employee can only work one shift per day.

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, $j \in \{1,...,7|J|\}$.
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. A nurse may not cover more than one shift each day.

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

The objectives are formulated as (weighted w_i) goals. The overall objective function is:

$$\text{Min}\, \overline{G}(x) = \sum_{i=1}^{16} w_i \overline{g}_i(x),$$

Where the goals are:

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

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

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.

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

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).

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

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.

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

12. A minimum of two consecutive night shifts.

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

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.

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

14. Maximum number of consecutive night shifts.

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

15. Maximum number of consecutive working days.

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

16. Shift cover requirements

$$\bar{g}_{16}(x) = \sum_{j=1}^{7|J|} \sum_{k \in K} \left| \sum_{i \in I} x_{ijk} - d_{jk} \right|$$

2. Solution Initialisation

In the HyFlex framework it is necessary to implement a method for initialising a new solution. For the personnel scheduling domain the solution is initialised using local search heuristic 3 described in section 3. The pseudocode for this heuristic is shown in Figure 9. It is a hill climbing heuristic which uses a neighbourhood operator which adds new shifts to the roster.

3. Heuristics

In HyFlex, the heuristics are classified into four categories:

Mutation : A heuristic which randomly mutates a solution. This usually returns a solution which is worse than the original solution.

Crossover : A heuristic which combines two solutions to produce a new solution. Usually aiming to keep some of the good qualities of both parent solutions in the new solution.

Ruin and recreate : A perturbative heuristic which changes part of a solution (usually making the solution worse) and then attempts to recreate/repair it.

Local search : A heuristic which attempts to improve the objective function value of the solution it is applied to. The new solution will not have a worse objective function value but it may be the same as the original solution's value.

There are also two general parameters which can be set (in the range 0.0 to 1.0) by the hyperheuristic. They are *Depth of search* and *Intensity of mutation*. One or both of these parameters may have an effect on each heuristic.

We will now describe the heuristics in each category currently implemented in the personnel scheduling domain.

Mutation Heuristic

Mutation heuristic 1: randomly un-assigns a number of shifts, keeping a feasible solution. The number of shifts the operator un-assigns is proportional to the intensity of mutation parameter.

Crossover Heuristics

There are currently three crossover heuristics.

Crossover heuristic 1 was presented in [3]. It operates by identifying the best *x* assignments in each parent and making these assignments in the offspring. The best assignments are identified by measuring the change in objective function when each shift is temporarily unassigned in the roster. The best assignments are those that cause the largest increase in the objective function value when they are unassigned. The parameter *x* ranges from 4-20 and is calculated using the *intensity of mutation* parameter as below:

```
x = 4 + round((1 - intensityOfMutation) * 16)
```

Crossover heuristic 2 was published in [5]. It creates a new roster by using all the assignments made in the parents. It makes those that are common to both parents first and then alternately selects an assignment from each parent and makes it in the offspring unless the cover objective is already satisfied.

Crossover heuristic 3 creates the new roster by making assignments which are only common to both parents.

Local Search Heuristics

A number of neighbourhood operators have been previously proposed for the personnel scheduling problem. They can be classified into three groups: *Vertical* swaps, *Horizontal* swaps and *New* swaps.

The *New* swaps are so called because they introduce new shifts into the roster (or oppositely delete shifts). Examples of these swaps are given in Figure 1 and Figure 2.

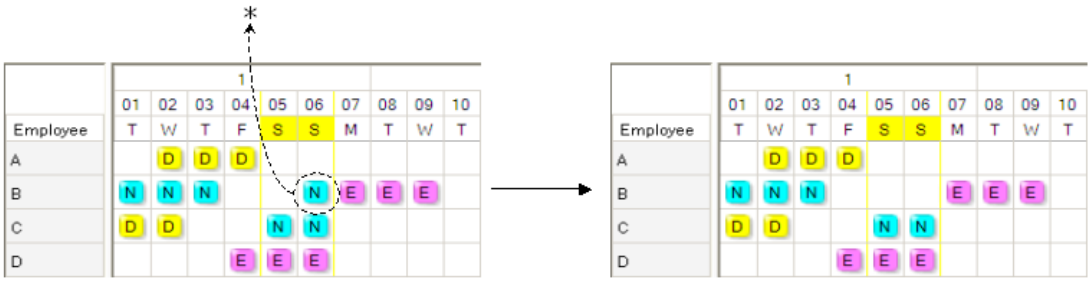

Figure 1

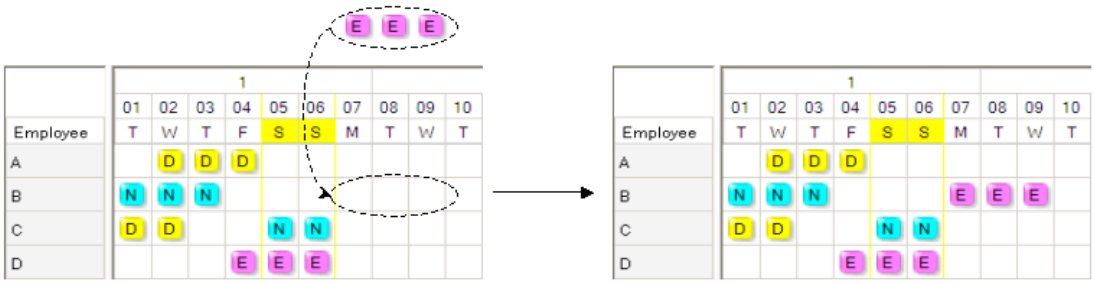

Figure 2

Horizontal swaps move shifts in single employee's work pattern hence the shifts move horizontally in the roster. Examples are given in Figure 3 and Figure 4.

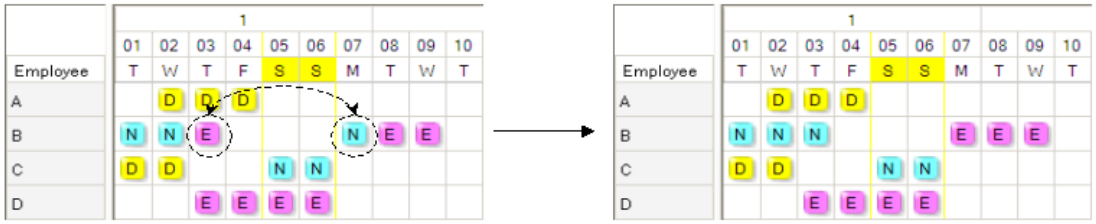

Figure 3

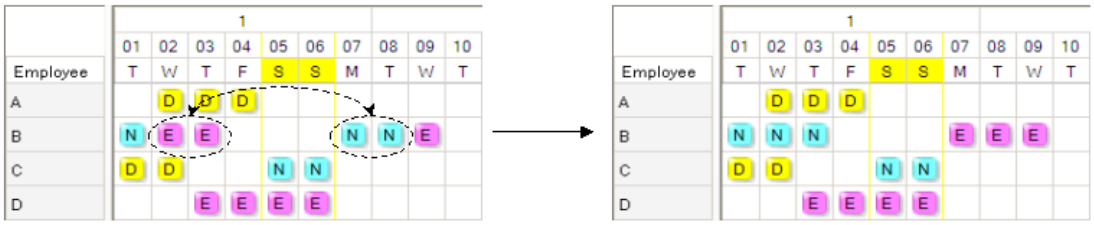

Figure 4

Vertical swaps move shifts between two employees hence the shifts move vertically in the roster. Examples are given in Figure 5 and Figure 6.

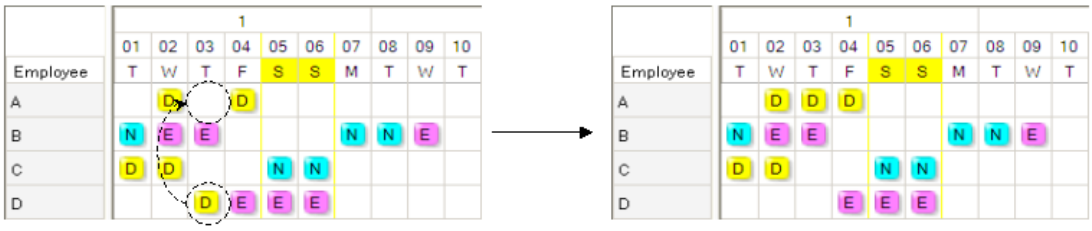

Figure 5

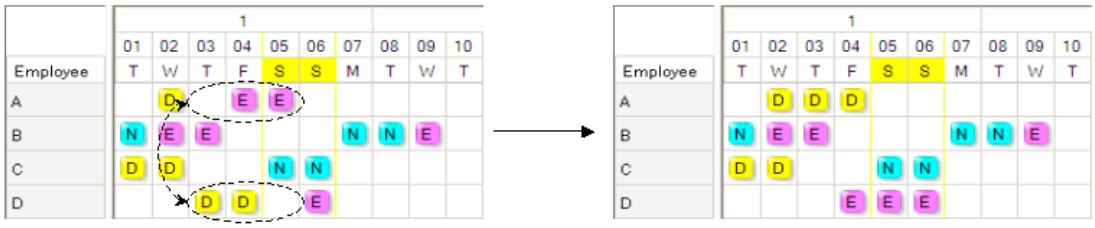

Figure 6

For all of these neighbourhood operators it is possible to move/swap shifts over a *block* of adjacent days. For example the swaps/moves shown in Figure 1, Figure 3 and Figure 5 can be considered moves over a block of days of length one day. Figure 2 and Figure 6 show moves over blocks of length three days and Figure 4 illustrates a swap using blocks of length two days. The length of the block to test is a parameter which can be set for each neighbourhood operator.

Local search heuristics 1-3 are 'hill climbers' which each use one of these types of neighbourhood operator. Note that, technically they are actually hill descenders as we are minimising the objective function, but we will use the term hill climber as it tends to be more familiar. Heuristic 1 uses the *vertical* neighbourhood operator, heuristic 2 uses the *horizontal* neighbourhood operator and heuristic 3 uses the *new* neighbourhood operator. The pseudocode for these heuristics is shown in Figure 7, Figure 8 and Figure 9 respectively. For each heuristic, the moves/swaps that are tested range in size of block length one up to a maximum block length (*MaxBlockLength*). This parameter is set to a value of 5. Previous investigations suggest that this is a good choice with regard to the balance between solution quality and computation time (see for example [6]).

```
1.  WHILE there are untried swaps
2.    FOR BlockLength=1 up to MaxBlockLength
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 7 Pseudocode for 'vertical' hill climber

```
1.  WHILE there are untried swaps
2.    FOR BlockLength=1 up to MaxBlockLength
3.      FOR each employee (E1) in the roster
4.        FOR each day (D1) in the planning period
5.          FOR D2 := D1+BlockLength, up to num days in planning period
6.            Swap all assignments for E1 on D1 up to D1+BlockLength
              with all assignments for E1 on D2 up to D2+BlockLength
7.            IF an improvement in roster penalty THEN
8.              Break from this loop and move on to the next day (D1+1)
9.            ELSE
10.             Reverse the swap
11.           ENDIF
12.         ENDFOR
13.       ENDFOR
14.     ENDFOR
15.   ENDFOR
16. ENDWHILE
```

Figure 8 Pseudocode for 'horizontal' hill climber

```
1.  WHILE there are untried swaps
2.    FOR BlockLength=1 up to MaxBlockLength
3.      FOR each employee (E1) in the roster
4.        FOR each day (D1) in the planning period
5.          FOR each shift type (S1) (including day off)
6.            Remove all assignments for E1 on D1 up to D1+BlockLength
              and assign shifts of type S1 to E1 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 (D1+1)
9.            ELSE
10.             Reverse the swap
11.           ENDIF
12.         ENDFOR
13.       ENDFOR
14.     ENDFOR
16.   ENDFOR
17. ENDWHILE
```

Figure 9 Pseudocode for 'new' shifts hill climber

Local search heuristics 4 and 5 are variants of the variable depth search described in [6]. The first version is similar to the one presented in that paper. The main difference is that it also uses *new* moves such as the ones shown in Figure 1 and Figure 2 as links in the ejection chain (the original version only tests *vertical* swaps as it was originally designed for instances where the cover requirements were a hard constraint rather than an objective). In the second version, as well as using the moves shown in Figure 1 and Figure 2, as potential links in the ejection chains, it also tests replacing an entire work pattern for a single employee as a link in the chain. These patterns are generated using a greedy heuristic method. The maximum search time for these two heuristics is set as: the *depth of search* parameter multiplied by a maximum time of 5 seconds.

'Ruin and Recreate' Heuristics

The ruin and recreate heuristics implemented are based on the one presented in [4].

Ruin and Recreate Heuristic 1: The heuristic works by un-assigning all the shifts in one or more randomly selected employees' schedules before heuristically rebuilding them. They are rebuilt by firstly satisfying objectives related to requests to work certain days or shifts and then by satisfying objectives related to weekends. For example min/max weekends on/off, min/max consecutive working or non working weekends, both days of the weekend on or off etc. Other shifts are then added to the employee's schedule in a greedy fashion, attempting to satisfy the rest of the objectives. Finally, shifts in this schedule are then swapped/moved by hill climber heuristics which use the *horizontal* and *new* neighbourhood operators.
In [4] it was observed that it was best to un-assign and rebuild only 2-6 work patterns at a time (for instances of all sizes). For this reason the first ruin and recreate heuristic un-assigns x schedules where x is calculated using the *intensity of mutation* parameter as follows:

$$x = \text{Round}(intensityOfMutation * 4) + 2$$

Ruin and Recreate Heuristic 2: The second heuristic provides a larger change to the solution by setting x using:

$$x = \text{Round}(intensityOfMutation * \text{Number of employees in roster})$$

Ruin and Recreate Heuristic 3: The third variant of the heuristic creates a small perturbation in the solution by using x=1.

4. Instances

The instances have been collected from a number of sources. Some of the instances are from industrial collaborators. These include: ORTEC an international consultancy and software company who specialise in workforce planning solutions and SINTEF, the largest independent research organisation in Scandinavia. Other instances have been provided by other researchers or taken from various publications. The collection is a very diverse data set drawn from eleven different countries. The majority of the instances are real world scenarios. Table 1 lists the instances. As can be seen, they vary in the length of the planning horizon, the number of employees and the number of shift types. Each instance also varies in the number and priority of objectives present.

	Instance	Best known	Staff	Shift types	Length (days)	Ref.
	BCV-1.8.1	252	8	5	28	[2, 7]
	BCV-1.8.2	853	8	5	28	[2, 7]
	BCV-1.8.3	232	8	5	28	[2, 7]
	BCV-1.8.4	291	8	5	28	[2, 7]
	BCV-2.46.1	**1572**	46	4	28	[2, 7]
	BCV-3.46.1	3280	46	3	26	[2, 7]
	BCV-3.46.2	**894**	46	3	26	[2, 7]

	BCV-4.13.1	**10**	13	4	29	[2, 7]
	BCV-4.13.2	**10**	13	4	28	[2, 7]
	BCV-5.4.1	**48**	4	4	28	[2, 7]
	BCV-6.13.1	768	13	5	30	[2, 7]
	BCV-6.13.2	392	13	5	30	[2, 7]
	BCV-7.10.1	**381**	10	6	28	[2, 7]
	BCV-8.13.1	**148**	13	5	28	[2, 7]
	BCV-8.13.2	148	13	5	28	[2, 7]
	BCV-A.12.1	1294	12	5	31	[2, 7]
	BCV-A.12.2	1953	12	5	31	[2, 7]
	ORTEC01	**270**	16	4	31	[4]
	ORTEC02	**270**	16	4	31	[4]
	GPost	**5**	8	2	28	
	GPost-B	**3**	8	2	28	
	QMC-1	**13**	19	3	28	
	QMC-2	**29**	19	3	28	
	Ikegami-2Shift-DATA1	**0**	28	2	30	[9]
	Ikegami-3Shift-DATA1	**2**	25	3	30	[9]
	Ikegami-3Shift-DATA1.1	**3**	25	3	30	[9]
	Ikegami-3Shift-DATA1.2	**3**	25	3	30	[9]
	Millar-2Shift-DATA1	**0**	8	2	14	[9]
	Millar-2Shift-DATA1.1	**0**	8	2	14	[9]
	Valouxis-1	**20**	16	3	28	[13]
	WHPP	**5**	30	3	14	[14]
	LLR	**301**	27	3	7	[10]
	Musa	**175**	11	1	14	[11]
	Ozkarahan	**0**	14	2	7	[12]
	Azaiez	**0**	13	2	28	[1]
	SINTEF	**0**	24	5	21	
	CHILD-A2	1111	41	5	42	
	ERMGH-A	795	41	4	48	
	ERMGH-B	1459	41	4	48	
	ERRVH-A	2197	51	8	48	
	ERRVH-B	6859	51	8	48	
	MER-A	9915	54	12	48	

Table 1Problem Instances

The instances can be downloaded from http:///www.cs.nott.ac.uk/~tec/NRP/. New results, instances and other related information and software are also be published at this location.

5. Conclusion

We have described the implementation of a personnel scheduling problem domain for the hyperheuristic software framework HyFlex. Within this domain we have implemented a number of heuristics for this problem. These heuristics have appeared in various publications on personnel scheduling and have be shown to be successful methods. The benchmark instances available are diverse and challenging. The majority are real world and have been taken from scenarios worldwide.

References

1. 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.
2. Brucker, P., E.K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe, *A Shift Sequence Based Approach for Nurse Scheduling and a New Benchmark Dataset.* Journal of Heuristics, Accepted for publication, 2009.
3. 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.
4. 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, 2008. **188**(2): pp. 330-341.
5. Burke, E.K., T. Curtois, R. Qu, and G. Vanden Berghe, *A Scatter Search for the Nurse Rostering Problem*. 2010, Journal of the Operational Research Society, **61**: pp. 1667-1679.
6. Burke, E.K., T. Curtois, R. Qu, and G. Vanden Berghe, *A Time Predefined Variable Depth Search for Nurse Rostering*. 2007, School of Computer Science and IT, University of Nottingham. Technical Report. Available from: http://www.cs.nott.ac.uk/TR/2007/2007-6.pdf
7. Burke, E.K., T. Curtois, R. Qu, and G. Vanden Berghe. *Problem Model for Nurse Rostering Benchmark Instances*. 2008; Available from: http://www.cs.nott.ac.uk/~tec/NRP/papers/ANROM.pdf.
8. Burke, E.K., J. Li, and R. Qu, *A Hybrid Model of Integer Programming and Variable Neighbourhood Search for Highly-constrained Nurses Rostering Problems.* European Journal of Operational Research, 2008 (accepted for publication).
9. Ikegami, A. and A. Niwa, *A Subproblem-centric Model and Approach to the Nurse Scheduling Problem.* Mathematical Programming, 2003. **97**(3): pp. 517-541.

10. 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.
11. Musa, A. and U. Saxena, *Scheduling nurses using goal-programming techniques.* IIE transactions, 1984. **16**: pp. 216-221.
12. Ozkarahan, I. *The Zero-One Goal Programming Model of a Flexible Nurse Scheduling Support System, in Proceedings of International Industrial Engineering Conference*. in Proceedings of International Industrial Engineering Conference. 1989. pp. 436-441.
13. 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.
14. Weil, G., K. Heus, P. Francois, and M. Poujade, *Constraint programming for nurse scheduling.* IEEE Engineering in Medicine and Biology Magazine, 1995. **14**(4): pp. 417-422.