Journal of the Operational Research Society (2011) 62, 360–367

www.palgrave-journals.com/jors/

Progress control in iterated local search for nurse rostering

EK Burke[1], T Curtois[1], LF van Draat[2], J-K van Ommeren[3] and G Post[2,3*]
[1]*University of Nottingham, Nottingham, UK;* [2]*ORTEC bv, Gouda, the Netherlands; and* [3]*University of Twente, Enschede, the Netherlands*

This paper describes an approach in which a local search technique is alternated with a process which 'jumps' to another point in the search space. After each 'jump' a (time-intensive) local search is used to obtain a new local optimum. The focus of the paper is in monitoring the progress of this technique on a set of real world nurse rostering problems. We propose a model for estimating the quality of this new local optimum. We can then decide whether to end the local search based on the predicted quality. The fact that we avoid searching these bad neighbourhoods enables us to reach better solutions in the same amount of time. We evaluate the approach on five highly constrained problems in nurse rostering. These problems represent complex and challenging real world rostering situations and the approach described here has been developed during a commercial implementation project by ORTEC bv.

Journal of the Operational Research Society (2011) **62,** 360–367. doi:10.1057/jors.2010.86
Published online 28 July 2010

Keywords: local search; iterated local search; nurse rostering

1. Introduction

Search methods which are focussed upon intensification and diversification have been widely discussed in the context of metaheuristic development (Voss *et al*, 1999; Glover and Kochenberger, 2003). Intensification corresponds, in some sense, to local search; the neighbourhood of a solution is searched *intensively* for solutions which are better or have better opportunities. On the other hand, with diversification the goal is to escape from relatively small neighbourhoods to solutions which may lead to better final results. A heuristic that is motivated by the trade off between intensification and diversification, is the Iterated Local Search approach. It was applied to several problems (see, for example, Stützle, 1998, 2006; Paquete and Stützle, 2002). An overview of Iterated Local Search can be found in (Loureno *et al*, 2003). The main loop of the Iterated Local Search algorithm consists of a *Perturb-Improve-Evaluate* cycle. The Perturbation is sometimes called *Kick*, *Mutation* or *Restart*. The Improvement phase is a Local Optimisation which usually ends if the local optimum is reached. The evaluation is done using an *Acceptance Criterion*.

Here we focus on the Acceptance Criterion, which we also apply before the local optimisation ends: in some cases the *progress* of the local optimisation is correlated to the local optimum it will reach. This motivated us to introduce a model that tries to predict a lower bound for the quality of the local optimum that we will reach during local optimisation. This model is employed to identify 'bad' local optimisations, which are halted before the local optimum is reached. We apply our approach in the context of nurse rostering problems.

In Section 2, we discuss the nurse rostering problem. Section 3 introduces our Iterated Local Search method and presents a model for the progress. In Section 4, we describe the experiments we performed; and in Section 5, we describe the results for five data sets from the area of nurse rostering. Finally, in Section 6, we discuss the results and give some conclusions.

2. The nurse rostering problem

Automating the nurse rostering process removes a frustrating chore and results in higher quality rosters which reduce hospital costs and increase employee satisfaction. Constructing nurse rosters is however a far from trivial problem (Lau, 1996). Owing to this complexity and its practical relevance the nurse rostering problem has attracted a large amount of research and a wide variety of approaches have been used to provide solutions. These methods include mathematical programming (especially column generation-based techniques) (Warner, 1976; Thornton and Sattar, 1997; Jaumard *et al*, 1998; Mason and Smith, 1998; Millar and Kiragu, 1998; Eveborn and

*Correspondence: G Post, Department of Applied Mathematics, University of Twente, Enschede, AE 7500, the Netherlands.
E-mail: g.f.post@math.utwente.nl

Rönnqvist, 2004; Bard and Purnomo, 2005a, b, 2007), constraint programming (Darmoni *et al*, 1995; Weil *et al*, 1995; Cheng *et al*, 1997; Meisels *et al*, 1997; Abdennadher and Schlenker, 1999; Chun *et al*, 2000; auf'm Hofe, 2000), local search and metaheuristics (Dowsland 1998; Burke *et al*, 1999, 2001a, 2007; Schaerf and Meisels 1999; Ikegami and Niwa 2003; Bellanti *et al*, 2004; Aickelin and Li 2007), evolutionary algorithms (Tanomaru, 1995; Aickelin and Dowsland, 2000, 2003; Cai and Li, 2000; Jan *et al*, 2000; Burke *et al*, 2001b; Dias *et al*, 2003; Özcan, 2005), case-based reasoning (Beddoe, 2004; Beddoe and Petrovic, 2006, 2007) and hyperheuristics (Burke and Soubeiga, 2003; Burke *et al*, 2003). More are discussed in recent surveys (Burke *et al*, 2004; Ernst *et al*, 2004). Although many novel and effective algorithms have been published, it is not possible to identify a dominant optimisation method for nurse rostering. This can be partly explained by the simple, practical reason of a lack of real world benchmark data sets. Fortunately, this deficiency is gradually being corrected and a variety of instances are being compiled and shared (Curtois, 2008). The data sets we test with in Section 5. are drawn from this collection.

The nurse rostering problem can be challenging to model due to large numbers of constraints and complex objective functions. However, it is relatively simple to understand at a more basic level: a set of shifts must be assigned to a set of nurses over a fixed planning period subject to a set of constraints. The constraints are related to the quality of the nurses' work patterns and ensuring sufficient numbers of nurses are present at certain times of the day (coverage). The objective function, which can be regarded as a set of soft constraints with relative priorities, maximises the nurses' satisfaction with their schedules and/or minimises the hospital's costs through reducing the number of shifts assigned.

The data sets we use in Section 5. contain a range of soft constraints for each nurse such as:

- maximum numbers of weekend shifts;
- maximum numbers of night shifts;
- minimum numbers of consecutive days off and on;
- maximum numbers of consecutive days on or of certain shifts;
- minimum and maximum numbers of assignments or working hours per scheduling period and per week;
- undesirable shift rotations;
- ensuring either Saturday and Sunday are working days or neither are working days ('Complete Weekends');
- requests for days on/off or shift pre-assignments; and
- ensuring two nurses are working or are not working at the same time.

Formal definitions of these constraints can be found in (Vanden Berghe, 2002; Burke *et al*, 2008) their implementations are also provided in the source code publicly available at (Curtois, 2008). Coverage is a hard constraint and levels of cover must not deviate above or below a fixed level.

3. The algorithmic methodology

3.1. Iterated local search

The main motivation of the work described in this paper was to build an effective and simple approach for automatically building nurse rostering solutions to challenging real world problems that are faced by ORTEC's customers. This simple methodology can be overviewed as follows: After reaching a local optimum with respect to a given neighbourhood structure, we 'kick' the best solution found so far; so based on the best solution we jump to another point in the search space. From here, we again perform local search. If we reach a better local optimum, we take this new solution as the starting point for further investigation, otherwise we revert to the old local optimum, and continue from there with a new kick. The approach can be illustrated by the following pseudocode:

```
procedure ILS(varS*: Solution);
var S, S′ : Solution;
begin
1:   S* ← LocalOptimum(S*);
2:   while remaining time >0 do
3:    begin
4:    S ← KickOf (S*);
5:    S′ ← LocalOptimum(S );
6:   if Cost(S′ ) < Cost(S*) then
7:   S* ←S′ ;
8:   end;
end.
```

Algorithm 1. Iterated Local Search

When employing an iterated local search, it is normal that most of the calculation time is spent on local optimisation with respect to the neighbourhood; in our experiments (see Section 4) this time represents more than 99% of the computational time. A possibility to improve the performance of this method is to detect, somehow, that it is useless to continue the local optimisation in line 5 of Algorithm 1. A rather obvious way is to stop line 5 if, after some fixed amount of time, a certain quality is not reached. However, this is not satisfactory because of the uncertainty surrounding how much computational time will be required to reach that *quality* of solution.

In this paper, we propose a model for monitoring the progress of the local search process that is independent of the optimisation tool that is used. Our approach is to *underestimate* the expected cost of the local optimum we are moving to. Hence, in general, we are cautious in stopping the local optimisition process.

3.2. Modelling the progress

Suppose we are given a solution S, which is a result of applying Algorithm 1 to the solution S'. We ask ourselves the following question: What can be said about the intermediate results? The local search attempts a lot of small moves, and will execute a move if it decreases the cost. The local search ends when none of all possible moves reduces the cost. Let us assume that every τ milliseconds we get an update of the result. Define $t_n = n \cdot \tau$ for $n = 0, 1, 2, \ldots$. We denote by S_n the resulting schedule at time t_n, and by c_n the cost of S_n. In the notation of Algorithm 1, we have that $S_0 = S$, and $S_n = S'$ if n is big enough. Since we apply local optimisation, we know that $c_{n+1} \leqslant c_n$. We denote by c_∞ the cost of the local optimum that is reached, that is c_n for n large enough.

What can we expect about the time-dependency of the cost? To be able to analyse the situation, we assume that successful moves incur a cost reduction of one unit. The moves are selected randomly, hence it seems reasonable that the chance that a move at time t_n is successful is proportional to the difference $c_n - c_\infty$. Let λ the proportionality factor. For the expected value $E(c_n)$ of the cost c_n we obtain

$$E(c_n) - c_\infty = (1 - \lambda)(c_{n-1} - c_\infty).$$

Solving this recursion relation we obtain

$$E(c_n) = (c_0 - c)(1 - \lambda)^n + c_\infty. \tag{1}$$

Several assumptions (not valid in practice!) underlie this model:

1. The change of improvement is proportional to the difference of the current cost and the end cost. If in the set of all moves, there would be a unique move incurring an improvement, this could be a correct assumption. However, in practice this is not the case: easy violations of soft constraints can be removed by several moves. This implies that λ decreases with time, which indicates that our early estimates will be optimistic.
2. The cost change of a move is always the same. In practice this is not correct. Even if all weights are equal, a certain move could lift two violations (or more) in one move. Soft constraint violations with high costs are usually detected earlier, as a move can lift this high cost at introducing small costs for other soft constraints. Again this implies that early estimates will be optimistic.
3. All moves take an equal calculation time. In practice this is not exactly the case, but if τ is relatively big, several hundreds of moves will occur between t_n and t_{n+1}, and the number of these moves is with high probability close to average.

A better understanding of the improvement process is very hard to acquire, and not the purpose of this paper; our purpose is to have some correlation between the predicted progress and the realised progress. The realised progress is usually very irregular, especially in the beginning of the search process. In our opinion, it will be impossible to characterise the random variables for the highly constrained nurse rostering problems we consider.

By interpolation of formula (1) we get the continuous time formula:

$$c(t) = Ae^{Bt} + C, \tag{2}$$

The relations between the parameters in the formulae (1) and (2) are given by $A = c_0 - c_\infty$, $C = c_\infty$, and $e^{B\tau} = \lambda$. In the sequel the continuous time formula will be used with one more simplification: we will assume that $C = 0$, that is we assume that the search process is on the way to obtain a perfect schedule. Again this is an optimistic point of view. Based on this assumption and the reported progress, we estimate how long it will take before the search process will improve the current lowest cost c^* of schedule S^*. We fix a reference time T, and base our decision to stop the search process, on whether the expected time t^* to improve S^* is larger than this reference time T or not.

Let us describe our method in more detail. We assume the cost $c(t)$ develops according to the formula

$$c(t) = Ae^{Bt}. \tag{3}$$

If we know the progress up to time t_n (for some positive integer n), our first task is to estimate the parameters A and B. For this we take the (natural) logarithm of formula (3), and obtain the line

$$y = \alpha t + \beta, \tag{4}$$

with $y = \log c(t)$, $\alpha = B$ and $\beta = \log A$. From the progress we have points (t_k, y_k) for $k = 0, 1, 2, \ldots, n$ and $y_k = \log(c_k)$. For $n \geqslant 2$ we can use the least squares method to obtain α and β. More explicitly, α and β are determined by the linear equations

$$\alpha \sum_{k=0}^{n} t_k + \beta(n+1) = \sum_{k=0}^{n} y_k. \tag{5}$$

$$\alpha \sum_{k=0}^{n} t_k^2 + \beta \sum_{k=0}^{n} t_k = \sum_{k=0}^{n} t_k y_k. \tag{6}$$

Once we solved α and β from these equations, the parameters A and B are given by $\alpha = B$ and $\beta = \log A$.

Based upon (3), we now can calculate the time t^* such that $c(t^*) = c^*$ where c^* is the cost of S^*. For this we solve t^* from the equation

$$Ae^{Bt^*} = c^*.$$

If $B < 0$, there is a unique solution for t^*; if $B \geqslant 0$ we take $t^* = \infty$.

The decision to stop the search process only will depend on whether t^* exceeds the reference time T or not. How the decision is reached, is the subject of the next section.

3.3. Progress control

Above we described the problem that we want to consider, and the model we use for the progress. In this section we will describe the controller. We assumed that we have an estimate for the time T needed to complete the local optimisation. In practice we use for T the time spent in line 1 in Algorithm 1. Apart from this, we introduced the time-unit τ, such that $t_n = n\tau$; we assumed that for each t_n we have a measurement of the cost $c_n = c(n\tau)$. The progress control will use the sequence (c_n) as input; as time progresses, more and more elements of the sequence (c_n) become available. Based on this sequence we calculate t^*, the time we expect that is needed to reach a result better than S^*, and depending on whether $t^* > T$ or not, we base the decision to stop the local optimisation or not. If c_n is already below c^*, we continue, independently of t^*. For calculating t^* and taking this decision to stop we use three parameters p, f, and m. The parameters are

- (**p**) We use a *rolling horizon*: as time progresses, the cost changes from the beginning become less interesting. Therefore we decide to forget about early points, and estimate the progress only based in the last p points. The formulae in (5) are valid, but only the last p points are used.
- (**f**) We use a *start-up time*: in the beginning of the local optimisation, we record whether $t^* > T$ or not, but we allow the search process to proceed, independently of it. The number of periods free of checks is denoted by f. Hence the earliest moment that we can stop is t_f. If $f = \infty$ we perform ILS without progress control.
- (**m**) We use *forgiveness*: we do not necessarily stop directly if some $t_n^* > T$, but only if this happened more than m times in the last p checks (we use the same base of p points). If $m = 0$, we do stop as soon as $t^* > T$. Clearly we will never stop before t_m; hence it is useless to consider $f < m$. If $m \geqslant p$, we do not stop at all.

A combination of these parameters (p, f, m), we call a *strategy*. Our main objective is to investigate the influence of a chosen strategy on the quality of the solution, assuming that we have a limited time at our disposal.

4. Experimental set-up

4.1. Data

We use the Iterated Local Search as introduced in (Burke *et al*, 2008). The data sets BCV-A.12.1, BCV-1.8.1, and BCV-2.46.1 were scheduled with this ILS. A variant of this algorithm is implemented in the advanced planning software HARMONY, a tool developed by ORTEC for workforce management and scheduling. The data sets ORTEC01 and ORTEC02 were tested here. For a description of the data sets we refer to Curtois (2008). The data sets have the following characteristics:

Table 1 Characteristics of the data sets

Name	*Employees*	*Shifts*	*Best*	*Start*	τ	T/τ
BCV-A.12.1	12	202	1360	3345	10	775
BCV-1.8.1	8	140	252	319	10	77
BCV-2.46.1	46	616	1572	1698	10	1953
ORTEC01	16	288	285	3026	500	282
ORTEC02	16	288	945	4816	500	296

In Table 1, the column 'Employees' contains the number of employees in the data set, 'Shifts' the number of shifts to be assigned. 'Start' is the cost of the starting point of our experiments, and 'Best' the cost of the best solution known to us. Finally τ denotes the time span between t_n and t_{n+1}, and instead of T, we record T/τ, the number of times, that τ is contained in T.

For each of these five data sets, we have initial schedules with cost as stated in the column 'Start' above; this is S^* *after* line 1 in the terminology of Algorithm 1. From S^* we constructed $S_1, S_2, \ldots, S_{1000}$, which are the kicks of S^*. On these 1000 schedules S_i, we performed the local optimisation (line 5 in Algorithm 1), to obtain S'_i. During local optimisation, every τ milliseconds, the actual cost of S_i is recorded; hence for each local optimisation we have a cost sequence $(c_0, c_1, \ldots)$. All sequences are of finite length; the length of a sequence (the number of time units τ that local optimisation was performed) differs from sequence to sequence.

4.2. Experiments

For different strategies (p, f, m), we perform two types of experiments, which we call experiment A and experiment B; experiment A mimics the situation in which we use progress control, whereas experiment B investigates the effect of progress control on each of the schedules S_1, $S_2, \ldots, S_{1000}$ separately.

In experiment A, we select a permutation of S_1, $S_2, \ldots, S_{1000}$. On this permutation we apply the progress control strategy. To explain the method, let us assume that the selected permutation is

$$S_{76}, S_{932}, S_{283}, S_{777}, \ldots$$

The search process will start with schedule S_{76}, and at a certain moment the progress control strategy decides that continuing the local optimisation on S_{76} is not profitable. Therefore we stop the local optimisation and start with schedule S_{932}. Here the same happens, and the search switches to S_{283}. Although in the end S_{283} gives a schedule better than the best schedule S^*, the progress control strategy decides to stop and continue with S_{777}. Here finally an improvement is found. Hence, we need four (partial) local optimisations ('tries') before an improvement is found. For experiment A we record three numbers:

- *Time*: The time, measured in τ milliseconds, it takes to reach a schedule with cost lower than S^*.
- *Tries*: The number of (partial) local searches executed.
- *Decrease*: The decrease in cost of the found improvement.

We repeat experiment A for 1000 randomly selected permutations, and for each strategy we calculate the averages of the numbers above.

Experiment B simply runs the progress control strategy on S_1, S_2, ..., S_{1000}, and records the total time spent by the strategy, as well as the total decrease in cost found by the strategy. From this we calculate the improvement per time unit. We present the improvement ('Gain') in τ *seconds*, that is in 1000τ milliseconds.

4.3. Expectations

By changing one of the parameters of a strategy we can say that the strategy becomes either more or less stringent, which means that it will stop the local search earlier or later. Clearly, decreasing m or f makes a strategy more stringent. In general, decreasing p makes a strategy more stringent, as the progress does not benefit anymore from good results which were achieved in the past.

Applying progress control will result in stopping local search runs that lead to improvements. Although we lose time by stopping these 'good' local optimisations, we, of course, hope to gain time by stopping 'bad' local optimisations. The goal is that the time gain is bigger than the time loss, but this might change when the strategies become more and more stringent. So the average time needed to reach a better solution will probably decrease in the beginning and increase after some 'optimal' parameter value.

In addition, there is a side effect of applying our strategies. The local optimisations that were wrongly stopped are heuristically (probably) not strong. Therefore, we expect the average score of the improvements found by a more stringent strategy to be higher.

5. Results

In this section, we present extensive results for ORTEC02, and summarise the results for the other four data sets. We have chosen this data set, because it contains the least number of 'good' local optimisations, and applying progress control seems to be the most appropriate here. However in Section 5.2, we will see that the progress control in 'easier' situations is even more profitable.

5.1. Results for ORTEC02

To this data set, we applied the experiments as described in Subsection 4.2 for $p = 20, 25, 30, \ldots, 65$, $f = 4, 6, 8, 10, 12,$ and $m = 2, 3, \ldots, 10$. First, we present three tables that describe the average behaviour for one fixed parameter, while the other two range over all possibilities. Here, the columns 'Time', 'Tries', and 'Decrease' correspond to the items in Subsection 4.2. The last three columns correspond to the 'Gain' found in τ seconds: we give the minimal, average and maximal improvement among the varying parameters.

In Table 2, the number of points p is fixed to the values from 20 to 65, while f and m may vary. The row starting with '—' gives the results without progress control. Looking at the column 'Decrease', which is the decrease in cost after one successful run, we see that all averages are approximately 40% higher than running without progress control. These averages are remarkably close together; they vary from 447 to 458. Moreover, the time needed to reach this local optimum decreased from 945 without progress control to an average of 802 for $p = 35$, an increase of speed of 15%. In the last three columns, we see that the 'Gain' per $\tau = 500$ s increased from 343 without control, to 592 for the best parameter setting within $p = 30$. Note that for $p \neq 40$, the other parameters have a big influence on the minimal gain. For $p = 40$ we see that the minimal Gain is still 525. So even the worst setting for $p = 40$ gives an improvement in Gain of more than 50%!

In Table 3, we see a similar improvement in decrease of cost. We see that for $f \geqslant 8$ the minimal and average gains are the highest. Hence it seems to be a good idea to introduce such a free period. This reflects the fact that the local search is usually very irregular (with big jumps) in the beginning.

From Table 4, we cannot conclude much on how forgiving we should be; this table yields that the maximal value 592 is in the same cross section as 296, which is almost the lowest value. Moreover, raising m implicitly implies raising f, which explains why for big m the minimal Gains are quite high.

Table 2 Experiments for p on ORTEC02

p	*Time*	*Tries*	*Decrease*	*Gain*		
				Min	*Ave*	*Max*
—	945	4.8	324	—	343	—
20	869	18.4	457	292	514	568
25	831	16.0	458	328	539	583
30	810	14.4	457	360	551	592*
35	802	13.3	452	361	543	584
40	817	12.8	449	525	528	571
45	834	12.4	447	374	515	554
50	856	12.2	453	361	506	545
55	873	11.9	454	348	496	527
60	878	11.6	458	338	495	528
65	891	11.3	457	343	487	521

Table 3 Experiments for f on ORTEC02

f	*Time*	*Tries*	*Decrease*	*Gain*		
				Min	*Ave*	*Max*
—	945	4.8	324	—	343	—
4	943	18.9	447	292	472	588
6	891	14.9	465	388	506	588
8	818	12.1	465	485	541	590
10	815	11.4	463	485	539	592*
12	763	9.9	431	478	529	580

Table 4 Experiments for m on ORTEC02

m	*Time*	*Tries*	*Decrease*	*Gain*		
				Min	*Ave*	*Max*
—	945	4.8	324	—	343	—
2	906	17.9	447	296	489	592*
3	906	17.7	448	292	489	584
4	886	15.5	451	334	496	584
5	871	14.2	459	420	510	575
6	826	12.4	466	491	535	573
7	808	11.5	461	488	541	588
8	805	11.0	462	486	538	584
9	815	10.8	460	482	531	577
10	792	10.0	441	478	527	568

From the tables, we can conclude that the highest Gain for the considered parameters is obtained for $p = 30$, $f = 10$, and $m = 2$ (see the * items); in this case the Gain per 500 s is 592. For these parameters, we obtain (experiment A):

$$\text{Time} = 786.2,\ \text{Tries} = 13.3,\ \text{Decrease} = 482.$$

The cost decrease in experiment A over these 1000 experiments is 481 800, and the total time needed was 786.2×500 s. The improvement per 500 s is therefore $481\,800/786.2 = 613$ for experiment A, while the improvement per 500 s is 592 (the Gain) for experiment B.

5.2. Results for all data sets

For the other data sets, we performed similar tests. However, as opposed to the previous results, we will use a set-up time here: starting a new run costs 0.5% of the expected calculation time T. This is consistent with what we explained before: performing a kick costs less than 1% of the total calculation time. In this way, we prevent strategies from picking out the very best local optimisations for free. Moreover the improvement in τ seconds will be calculated from experiment A, as it gives a better indication for the number of times that the set-up should be accounted for. So, the Gain presented in Table 5 is the total decrease in cost (column 8), divided by the total time spent (column 5), multiplied by 1000 to switch from milliseconds to seconds. We present the results for the best found strategy, and add four more columns, which describe how often the best strategy takes the correct decision. Here 'Good'/'Bad' means that the local optimisation reaches/does not reach a better optimum, and 'stop', respectively 'cont.' indicates whether the strategy allowed the local optimisation to run till the end, respectively not till the end.

Several remarks can be made about these results. The first remark is that progress control is useful to undertake in all cases. The increase in Gain ranges from 35% for ORTEC01 to almost 300% for BCV-2.46.1 and BCV-1.8.1. The second remark is that the harder the case, the more forgiving we should be. The period free of check (f) should in general not be very high. Only ORTEC02 waits 9 time units before doing a first selection; at t_9 310 out of 1000 tries are stopped. In this respect BCV-2.46.1 is rather extreme; although the average time for a local optimisation is 1612 time units, 764 out of 1000 tries are stopped at t_3.

Note that in BCV-2.46.1 an average of 21 local optimisations are stopped, before the local optimisation is continued till the end to obtain the new local optimum.

6. Discussion and conclusions

From the results in Table 5, we can see that progress control is a good thing to do: looking at the 'Gain' in Table 5, we see improvements for the data sets of 101, 187, 195, 35, and 82%, respectively. When we implement this approach in practice, several questions arise. We consider some of these questions here.

- *How to obtain T?*
  In Algorithm 1, we start with a result, which is not yet locally optimal. We assume that the first local optimisation in line 1 is a good indication for the time needed to reach local optimality in the later attempts.[1] However, the method is not very sensitive for the exact value of T. Sometimes it is better to decrease T. For example, decreasing T for BCV-1.8.1 from 77 to 6 increases the Gain from 130 to 159.
- *How to obtain τ?*
  Applying progress control takes little time. Applying it 300 times in an average run (like in the ORTEC data sets) costs far less than 1% of the time required. Smaller timescales, like in the data set BCV-2.46.1, open the possibility of discarding many runs in very short time. Beforehand it is unclear whether this can have a positive effect; can we say something about a local optimisation if only 1 or 2% of it was executed? We think that the employment of our optimistic estimates leads us to an answer of yes; if the progress cannot reach our not so

[1] In the cases we studied T defined in this way is ample. This is in agreement with our idea that the progress control should be optimistic.

Table 5 Best found strategies

Name	*P**	*f**	*m**	*Time*	*Tries*	*Start*	*Decrease*	*Gain*	*Good stop*	*Good cont.*	*Bad stop*	*Bad cont.*
BCV-A.12.1	—	—	—	1090	1.5	3345	243	223	0	682	0	318
	14	3	0	948	10.1	3345	426	449	586	96	315	3
BCV-2.46.1	—	—	—	3105	1.9	1698	14	4.6	0	524	0	476
	33	3	1	2122	22	1698	28	13.2	478	46	470	6
BCV-1.8.1	—	—	—	245	2.7	319	10.9	44	0	361	0	639
	3	2	0	112	8.2	319	14.6	130	239	122	639	0
ORTEC01	—	—	—	515	3.4	3026	503	975	0	296	0	704
	46	3	2	459	14.7	3026	605	1319	229	67	646	58
ORTEC02	—	—	—	952	4.8	4816	324	341	0	200	0	800
	33	9	4	760	11.9	4816	472	622	116	84	748	52

severe limits in the beginning, it is better to stop right away. Whether we have to look after 2, 3, or 9 time units seems to be quite arbitrary.

- *How to obtain a good strategy?*
  Obtaining the optimal strategy will be very hard. However, good strategies are not so hard to find. Tables 2–4 are meant to indicate this; any reasonable strategy will be better than doing nothing. Looking at the columns 'Good cont.' and 'Bad cont.' in Table 5, we can understand why progress control is so successful. Of the local searches that reach the end, the percentages of good local optimisations for the five data sets are 97, 88, 100, 58, and 62%.
  These percentages are a clear improvement. This, paired to stopping many tries after a very short time, explains the benefits of progress control.
  The parameter settings to use can be obtained in a static way, or in a dynamic way. The static way means that the iterative local search is fed with some representative data sets, and the best parameters are determined beforehand, much in the way we did here.
  The dynamic method assumes that we will do many (say, at least 50) local optimisations. Then we could decide that out of every 100 tries, we perform the first 10 without progress control, and determine what would have been the optimal parameters. Note that in this case we also get a good indication for T. These parameters can be used for the next 90 attempts.

The problems we presented here are heavily constrained. Consequently, a kick can create a situation which the local optimiser cannot improve. We believe that this behaviour is one of the main reasons for the success of progress control presented here. This aspect of the iterated local search means that there is a strong correlation between the expected improvement time t^* and the quality of the end result.

It is clear that many refinements can be made to the basic idea of progress control as presented here. In the form it is now, we can imagine that it is also applicable to other types of heavily constrained problems where iterated local search is applied. For the customers of HARMONY a clear gain in performance was achieved with calculations times often significantly reduced.

References

Abdennadher S and Schlenker H (1999). Nurse scheduling using constraint logic programming. In: *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence*. AAAI Press: Menlo Park, CA, pp 838–843.

Aickelin U and Li J (2007). An estimation of distribution algorithm for nurse scheduling. *Ann Opns Res* **155**: 289–309.

Aickelin U and Dowsland KA (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *J Sched* **3**: 139–153.

Aickelin U and Dowsland KA (2003). An indirect genetic algorithm for a nurse scheduling problem. *Comput Opns Res* **31**: 761–778.

Bard JF and Purnomo HW (2005a). A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio Econ Plan Sci* **39**: 193–213.

Bard JF and Purnomo HW (2005b). Preference scheduling for nurses using column generation. *Eur J Opl Res* **164**: 510–534.

Bard JF and Purnomo HW (2007). Cyclic preference scheduling of nurses using a Lagrangian-based heuristic. *J Sched* **10**: 5–23.

Beddoe GR (2004). *Case-based reasoning in personnel rostering*. PhD thesis, University of Nottingham, UK.

Beddoe GR and Petrovic S (2006). Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *Eur J Opl Res* **175**: 649–671.

Beddoe GR and Petrovic S (2007). Enhancing case-based reasoning for personnel rostering with selected tabu search concepts. *J Opl Res Soc* **58**: 1586–1598.

Bellanti F, Carello G, Della Croce F and Tadei R (2004). A greedy-based neighborhood search approach to a nurse rostering problem. *Eur J Opl Res* **153**: 2840.

Burke EK and Soubeiga E (2003). Scheduling nurses using a tabu-search hyper-heuristic. In: Kendall G *et al* (eds). *1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)*. University of Nottingham, UK, pp 197–218.

Burke EK, De Causmaecker P and vanden Berghe G (1999). A hybrid tabu search algorithm for the nurse rostering problem. In: McKay B, Yao X, Newton CS, Kim J and Furuhashi T (eds) *Simulated Evolution and Learning, Selected Papers from the 2nd*

Asia-Pacific Conference on Simulated Evolution and Learning, SEAL 98. Lecture Notes in Artificial Intelligence, Vol. 1585, Springer: London, pp 187–194.

Burke EK, De Causmaecker P, Petrovic S and vanden Berghe G (2001a). Variable neighbourhood search for nurse rostering problems. In: Sousa JD (ed). *Proceedings of the 4th Metaheuristics Internation Conference (MIC 2001)*. Porto, Portugal, pp 755–760.

Burke EK, Cowling P, De Causmaecker P and vanden Berghe G (2001b). A memetic approach to the nurse rostering problem. *Appl Intell* **15**: 199–214.

Burke EK, Kendall G and Soubeiga E (2003). A tabu-search hyper-heuristic for timetabling and rostering. *J Heuristics* **9**: 451–470.

Burke EK, De Causmaecker P, vanden Berghe G and Van Landeghem H (2004). The state of the art of nurse rostering. *J Sched* **7**: 441–499.

Burke EK, Curtois T, Qu R and vanden Berge G (2007). *A time predefined variable depth search for nurse rostering*. Technical Report, School of Computer Science and IT, University of Nottingham, http://www.cs.nott.ac.uk/TR/2007/2007-6.pdf.

Burke EK, Curtois T, Post G, Qu R and Veltman B (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *Eur J Opl Res* **188**: 330–341.

Cai X and Li KN (2000). A genetic algorithm for scheduling staff of mixed skills under multi-criteria. *Eur J Opl Res* **125**: 359–369.

Cheng BMW, Lee JHM and Wu JCK (1997). A nurse rostering system using constraint programming and redundant modeling. *IEEE T Inf Technol B* **1**(1): 44–54.

Chun AHW, Chan SHC, Lam GPS, Tsang FMF, Wong J and Yeung DWM (2000). Nurse rostering at the hospital authority of Hong Kong. In: *Proceedings of the Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press/The MIT Press: Menlo Park, CA, pp 951–956.

Curtois T (2008). Nurse rostering benchmark data sets. http://www.cs.nott.ac.uk/~tec/NRP/.

Darmoni SJ, Fajner A, Mah N, Leforestier A, Von-dracek M, Stelian O and Baldenweck M (1995). HOROPLAN: Computer-assisted nurse scheduling using constraint-based programming. *J Soc Health Syst* **5**: 41–54.

Dias TM, Ferber DF, de Souza CC and Moura AV (2003). Constructing nurse schedules at large hospitals. *Int T Opl Res* **10**: 245–265.

Dowsland KA (1998). Nurse scheduling with tabu search and strategic oscillation. *Eur J Opl Res* **106**: 393–407.

Ernst AT, Jiang H, Krishnamoorthy M, Owens B and Sier D (2004). An annotated bibliography of personnel scheduling and rostering. *Ann Opns Res* **127**: 21–44.

Eveborn P and Rönnqvist M (2004). Scheduler—a system for staff planning. *Ann Opns Res* **128**: 21–45.

Glover FW and Kochenberger GA (eds). (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers: Boston, MA.

Ikegami A and Niwa A (2003). A subproblem-centric model and approach to the nurse scheduling problem. *Math Program* **97**: 517–541.

Jan A, Yamamoto M and Ohuchi A (2000). Evolutionary algorithms for nurse scheduling problem. In: *Proceedings of the 2000 Congress on Evolutionary Computation*. IEEE Press: California, USA, pp 196–203.

Jaumard B, Semet F and Vovor T (1998). A generalized linear programming model for nurse scheduling. *Eur J Opl Res* **107**: 1–18.

Lau HC (1996). On the complexity of manpower shift scheduling. *Comput Opns Res* **23**: 93–102.

Loureno HR, Martin OC and Stützle T (2003). Iterated local search. In: Glover F and Kochenberger G (eds). *Handbook of Metaheuristics*. Kluwer: Boston, MA, pp 321–353.

Mason AJ and Smith MC (1998). A nested column generator for solving rostering problems with integer programming. In: Caccetta L, Teo KL, Siew PF, Leung YH, Jennings LS and Rehbock V (eds). *International Conference on Optimisation: Techniques and Applications*. Perth, Australia, pp 827–834.

Meisels A, Gudes E and Solotorevsky G (1997). Combining rules and constraints for employee timetabling. *Int J Intell Syst* **12**: 419–439.

Meyer auf'm Hofe H (2000). Solving rostering tasks as constraint optimization. In: Burke EK and Erben W (eds). *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling*. Lecture Notes in Computer Science, Vol. 2079. Springer: Berlin, pp 191–212.

Millar HH and Kiragu M (1998). Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *Eur J Opl Res* **104**: 582–592.

Özcan E (2005). Memetic algorithms for nurse rostering. In: Yolum P *et al* (eds). *The 20th International Symposium on Computer and Information Sciences*. Springer-Verlag: Berlin, pp 482–492.

Paquete L and Stützle T (2002). An experimental investigation of iterated local search for coloring graphs. In: Cagnoni S *et al* (eds). *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops*. Springer-Verlag: London, UK, pp 122–131.

Schaerf A and Meisels A (1999). Solving employee timetabling problems by generalized local search. In: Lamma E and Mello P (eds). *Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence*. Springer-Verlag: Berlin, pp 380–389.

Stützle T (1998). *Applying iterated local search to the permutation flow shop problem*. Technical report, Technische Hochschule Darmstadt.

Stützle T (2006). Iterated local search for the quadratic assignment problem. *Eur J Opl Res* **174**: 1519–1539.

Tanomaru J (1995). Staff scheduling by a genetic algorithm with heuristic operators. In: *Proceedings of the IEEE Conference on Evolutionary Computation*. pp 456–461.

Thornton J and Sattar A (1997). Nurse rostering and integer programming revisited. In: Verma B and Yao X (eds). *International Conference on Computational Intelligence and Multimedia Applications*. Griffith University: Gold Coast, Australia, pp 49–58.

Vanden Berghe G (2002). *An advanced model and novel meta-heuristic solution methods to personnel scheduling in healthcare*. PhD thesis, University of Gent, Belgium.

Voss S, Martello S, Osman IH and Roucairol C (eds). (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers: Boston, MA.

Warner DM (1976). Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Opns Res* **24**: 842–856.

Weil G, Heus K, Francois P and Poujade M (1995). Constraint programming for nurse scheduling. *IEEE Eng Med Biol* **14**: 417–422.

Received June 2009;
accepted April 2010 after one revision