

A Hybrid Setup for a Hybrid Scenario: Combining Heuristics for the Home Health Care Problem

Stefan Bertels **Torsten Fahle**

University of Paderborn

Faculty of Computer Science, Electrical Engineering and Mathematics

Fürstenallee 11, D-33102 Paderborn, Germany

{bertels, tef}@uni-paderborn.de

Abstract

Home health care, i.e. visiting and nursing patients in their homes, is a growing sector in the medical service business. From a staff rostering point of view, the problem is to find a feasible working plan for all nurses that has to respect a variety of hard and soft constraints, and preferences. Additionally, home health care problems contain a routing component: A nurse must be able to visit her patients in a given roster using a car or public transport. It is desired to design rosters that consider both, the staff rostering and vehicle routing components while minimizing transportation costs and maximizing satisfaction of patients and nurses.

In this paper we present the core optimization components of the PARPAP software. In the optimization kernel, a combination of linear programming, constraint programming, and (meta-)heuristics for the home health care problem is used, and we show how to apply these different heuristics efficiently to solve home health care problems. The overall concept is able to adapt to various changes in the constraint structure, thus providing the flexibility needed in a generic tool for real-world settings.

1 Introduction

Home health care (HHC), i.e. visiting and nursing patients in their homes, is a growing sector in the medical service business. More and more private companies are now working in this area. As the nursing companies get larger, the problem of how to schedule the nursing staff arises. The challenge of this problem is to combine aspects of vehicle routing and staff rostering. Both are well known combinatorial optimization problems, and good algorithms for each of these two problems are known (see e.g. [24] and e.g. [2, 6, 9]). To obtain applicable solutions,

however, it is crucial to solve the nurse scheduling problem as a whole due to the high inter-dependencies of optimized routes and rostering constraints. Additionally, soft constraints and preferences require the use of specially designed algorithms. Rostering constraints include hard ones like qualification requirements or work time limitations and soft ones. Soft constraints are especially difficult to handle, but have to be considered for applicable schedules. Typical examples are:

- patients prefer certain time intervals for being served,
- the right “chemistry” between patients and staff has to be ensured,
- patients do not like frequent changes of nursing staff,
- staff satisfaction concerning e.g. work load and work time should be maximized.

The vehicle routing aspect of the problem has to take travel times and distances, and inhomogeneous fleets (bicycles, public transport, cars) into account. Time windows are important in both, the rostering, as well as the routing part of the home health care problem.

Solutions with minimal costs and maximal patients/staff satisfaction are of most interest for companies. Costs in this context may reflect expenses for fuel, etc. or costs of the staff.

Respecting preferences is the second important parameter to be considered in HHC rostering. Patients will simply change to a different health care company if their wishes are not met. For the staff, considering their preferences increases motivation, which on the one hand impacts on the patients, and on the other hand helps to deal with many stressful situations (death, terminal illnesses, late jobs, etc.).

An important real-world requirement in our project PARPAP is runtime limitation. Planners like to have a good solution after at most 10–15 minutes runtime. From our experience in crew rostering [9] this excludes column generation approaches for the HHCP. Instead, we will apply Tabu Search, Simulated Annealing, or Constraint Programming, respectively, to assign staff to jobs. For optimizing an individual work-plan, we will use a hybrid linear and constraint programming module.

In Sec. 2 we will describe how we model the HHC problem. For this presentation, a more compact model than the one in the industrial prototype will be used. The simplified model still reflects the key characteristics of the original problem, but allows us to ignore certain technical details when presenting the algorithms. Section 3 is dedicated to the heuristics developed for the problem, and we will show how to combine these methods in a powerful hybrid approach in Sec. 4. Section 5 presents numerical results of the various methods, and finally we conclude.

1.1 Literature Review

To our knowledge, there are only a few publications on the topic of optimization and scheduling in HHC: Cheng and Rich describe a combined mixed-integer programming (MIP) and heuristics approach [8]. Numerical results for up-to 4 nurses and 10 patients are presented. In [3] a decision support system that is based on simple scheduling heuristics is proposed.

Two related topics have attracted more researchers: Planning systems for hospitals model some aspects that are also needed for HHC. We only mention [1, 5, 7, 18] as examples. Most of these use constraint programming techniques in order to model and solve the nurse rostering problem. Vehicle Routing with Time Windows (see e.g. [24]) reflects the mobility aspect of the problem, but ignores any further restriction.

2 Mathematical Model for the HHCP

Staff rostering describes the process of assigning staff to tasks. In HHC, the staff consists of employers with various skills, and the tasks are the services to be provided to the patients, training, etc. Timing rules, qualification rules, relationship rules, and structural rules, as well as routing information dictate the way, a “good” roster should look. The “cost” of a roster depends on real costs for wages and transportation, as well as on artificial costs introduced as penalty terms for modeling certain characteristics and preferences.

2.1 Parameters and Notation

We are given N nurses $\mathcal{N} = \{1, \dots, N\}$, P patients $\mathcal{P} = \{1, \dots, P\}$, and a set of J jobs $\mathcal{J} = \{1, \dots, J\}$. Jobs represent either a certain service to be provided to a certain patient, or a task to be performed by a nurse during her working hours (like training, breaks, emergency service, etc.). Thus, there is a fixed location for each of these jobs.

For each patient $p \in \mathcal{P}$ there is a subset $\mathcal{J}_p \subseteq \mathcal{J}$ such that these jobs involve patient p , and it holds: $\bigcup_{p \in \mathcal{P}} \mathcal{J}_p \subseteq \mathcal{J}$, and $\mathcal{J}_p \cap \mathcal{J}_q \neq \emptyset \iff p = q$. Since jobs are patient related, we can always identify the patient corresponding to a job. In the following, we will only speak of jobs, and by *preferences of a job* we refer to the preferences of the patient corresponding to the job.

A *time window* represents the time interval in which a job has to be started, or describes the working time interval of a nurse. We distinguish between *soft* time windows and *hard* time windows. Whereas a soft time window is only a preference, which we may violate at the expense of penalty costs, any hard time window has to be met. Waiting time before a hard time window is allowed, beginning service after the hard time window leads to an infeasible roster. The following functions represent the hard and soft time windows of nurses and jobs, respectively:

- $hb_j : \mathcal{J} \rightarrow \mathbb{R}$, $he_j : \mathcal{J} \rightarrow \mathbb{R}$, describe start and end, resp. of a job’s *hard* time window.
- $sb_j : \mathcal{J} \rightarrow \mathbb{R}$, $se_j : \mathcal{J} \rightarrow \mathbb{R}$, describe start and end, resp. of a job’s *soft* time window.
- $d_j : \mathcal{J} \rightarrow \mathbb{R}$ is the duration of the job, i.e. the time needed to complete a job.
- $hb_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$, $he_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$, $sb_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$, $se_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$, represent the start and end of the *hard* and *soft* time window of a nurse.
- $min_time_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$ and $max_time_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}$ give minimal and maximal working time of a nurse.

Obviously, for a job $j \in \mathcal{J}$ we have $hb_j(j) \leq sb_j(j) \leq se_j(j) \leq he_j(j)$. The same holds for time windows of nurses. Figure 1 shows an example for these time windows.

Qualifications are the skills possessed by a nurse, or required by a job, respectively. The concept of qualifications offers a rather flexible tool for modeling various characteristics of the HHC problem. Let S be a set of all qualifications:

- $quali_j : \mathcal{J} \rightarrow 2^S$ and $quali_{\mathcal{N}} : \mathcal{N} \rightarrow 2^S$ are the qualifications required for a job or possessed by a nurse, respectively.

Hard qualifications are those that are vitally required for the job and include e.g. graduation. *Soft constraints* embrace preferences of nurses and patients, and may be ignored at the expense of penalty costs. We model preferences of patients for certain nurses, preferences of nurses for

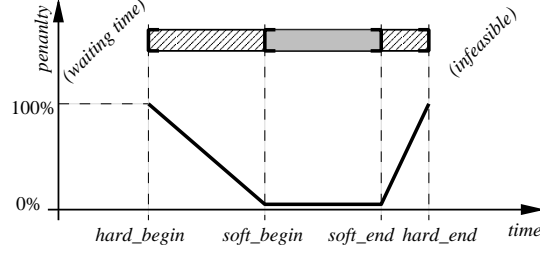


Figure 1: Penalty concept for time windows. Arriving before *hard_begin* produces waiting time, arriving after *hard_end* is infeasible. Penalties proportional to earliness or lateness are used for arrivals within the hard, but before or after the soft time window.

certain patients, experiences for certain jobs, and factors that guide a fair distribution of difficult jobs over all nurses (over some time period). Also, we allow soft qualifications and requests for those. In the compact model, all these parameters are accumulated into one function, which gives the soft constraint penalty value for assigning nurse n to job j :

- $sc : \mathcal{N} \times \mathcal{J} \rightarrow [0 \dots 1]$.

The last function provides geographical information. The *routing* aspect considers travel time between two jobs j, j' (in the industrial prototype travel distances are used as well). Since each job has a fixed location, we use the travel time between these locations:

- $tr_time : \mathcal{J} \times \mathcal{J} \rightarrow \mathbb{R}$, travel time between the locations of two jobs.

2.1.1 Core Optimization Problem

In the HHC problem we are looking for an assignment of job schedules to nurses, such that all jobs are taken care of, all hard constraints are respected, only few soft constraints are violated, and such that the overall cost for that assignment is minimal and the number of preferences satisfied is maximal. To be more specific, we formulate the following:

A sequence $R = [(j_1, t_1), \dots, (j_k, t_k)]$, $j_l \in \mathcal{J}, t_l \in \mathbb{R}, l = 1 \dots k$ is called a *roster* and contains k jobs and a starting time t_l for each job j_l . We assume that the sequence is ordered in increasing times, that is $t_l < t_{l+1}$ for $l = 1 \dots k - 1$. We need to find a *solution* $S = \{R^{(1)}, \dots, R^{(N)}\}$ consisting of N rosters, where $R^{(n)} = [(j_1^{(n)}, t_1^{(n)}), \dots, (j_{k_n}^{(n)}, t_{k_n}^{(n)})]$ is the roster for nurse n , such that:

$$\bigcup_{i=1}^N \bigcup_{l=1}^{k_i} \{j_l^{(i)}\} = \mathcal{J} \quad (1)$$

$$\forall 1 \leq i, i' \leq N, 1 \leq l \leq k_i, 1 \leq l' \leq k_{i'} : (j_l^{(i)} = j_{l'}^{(i')}) \Rightarrow (i = i' \wedge l = l') \quad (2)$$

$$hb_{\mathcal{J}}(j_l^{(i)}) \leq t_l^{(i)} \leq he_{\mathcal{J}}(j_l^{(i)}), \quad i = 1 \dots N, l = 1 \dots k_i \quad (3)$$

$$t_l^{(i)} + d_{\mathcal{J}}(j_l^{(i)}) + tr_time(j_l^{(i)}, j_{l+1}^{(i)}) \leq t_{l+1}^{(i)}, \quad i = 1 \dots N, l = 1 \dots k_i - 1 \quad (4)$$

$$\min_time_{\mathcal{N}}(i) \leq t_{k_i}^{(i)} + d_{\mathcal{J}}(j_{k_i}^{(i)}) - t_1^{(i)}, \quad i = 1 \dots N \quad (5)$$

$$\max_time_{\mathcal{N}}(i) \geq t_{k_i}^{(i)} + d_{\mathcal{J}}(j_{k_i}^{(i)}) - t_1^{(i)}, \quad i = 1 \dots N \quad (6)$$

$$\text{hb}_{\mathcal{N}}(i) \leq t_1^{(i)} \quad \text{and} \quad t_{k_i}^{(i)} + d_g(j_{k_i}^{(i)}) \leq \text{he}_{\mathcal{N}}(i). \quad (7)$$

$$\text{quali}_g(j_l^{(i)}) \subseteq \text{quali}_{\mathcal{N}}(i), \quad i = 1 \dots N, l = 1 \dots k_i \quad (8)$$

These hard constraints can be explained as follows: We need to cover all jobs by our schedules (1), but none of them more than once (2). Any starting point for a job has to respect the hard time window (3). In (4) we require enough time to provide the service and to travel to the next job before the next job starts. (5) and (6) define lower and upper bounds on the working time, and (7) ensures that no job is carried out outside the work time interval of the nurse. Finally, we insist on having all hard qualifications of a job covered by the assigned nurse (8).

2.1.2 Cost Function

In order to model the cost function, we have to define our measure for violating soft constraints. Any violation of a soft time window will be penalized by a factor proportional to the earliness or lateness. For all jobs $j \in \mathcal{J}$ let $t_j \in [\text{hb}_g(j), \text{he}_g(j)]$ be the time assigned to job j . Then

$$\text{early}_g(j) := \begin{cases} 0, & \text{sb}_g(j) = \text{hb}_g(j) \\ \frac{\text{sb}_g(j) - t_j}{\text{sb}_g(j) - \text{hb}_g(j)}, & \text{else} \end{cases} \quad (9)$$

$$\text{late}_g(j) := \begin{cases} 0, & \text{he}_g(j) = \text{se}_g(j) \\ \frac{t_j - \text{se}_g(j)}{\text{he}_g(j) - \text{se}_g(j)}, & \text{else} \end{cases} \quad (10)$$

$$p_g(j) = \max\{\text{early}_g(j), \text{late}_g(j), 0\} \quad (11)$$

Since t_j is within the soft time window ($\Rightarrow \text{early}_g(j) \leq 0$ and $\text{late}_g(j) \leq 0$), before the soft time window ($\Rightarrow \text{early}_g(j) > 0$ and $\text{late}_g(j) \leq 0$), or after it ($\Rightarrow \text{early}_g(j) \leq 0$ and $\text{late}_g(j) > 0$), choosing $p_g(j)$ as in (11) gives the correct penalty (see Fig. 1).

Violating nurses' soft time window is treated similarly. The corresponding penalty function is $p_{\mathcal{N}}(\cdot)$.

Let $R = [(j_1, t_1), \dots, (j_k, t_k)]$, $j_l \in \mathcal{J}, t_l \in \mathbb{R}, l = 1 \dots k$, be a roster assigned to nurse n . Any violated soft qualification is penalized by adding extra costs stemming from assigning these jobs to nurse n , and normalizing them by the number of jobs assigned:

$$p_{\text{sc}}(n) := \frac{1}{k} \cdot \sum_{l=1}^k \text{sc}(n, j_k) \quad n = 1 \dots N \quad (12)$$

Having defined a solution of the HHC problem and knowing the measure for soft constraints, we are now able to construct the objective function. In order to reflect both, the routing and the rostering aspects, we combine them into a weighted sum of the total travel time needed for the schedule and the sum of all penalties:

$$\begin{aligned} & \text{minimize} \quad \text{obj}(R^{(1)}, \dots, R^{(N)}), \quad \text{where} \\ \text{obj}(R^{(1)}, \dots, R^{(N)}) &= \alpha_1 \cdot \frac{\sum_{i=1}^n \sum_{l=1}^{k_i-1} \text{tr_time}(j_l^{(i)}, j_{l+1}^{(i)}) - LB_{\text{traveltime}}}{UB_{\text{traveltime}} - LB_{\text{traveltime}} + \varepsilon} \end{aligned} \quad (13)$$

$$+ \alpha_2 \cdot \frac{1}{N} \frac{\sum_{n \in \mathcal{N}} \left[t_{k_n}^{(n)} + d_g(j_{k_n}^{(n)}) - t_1^{(n)} \right] - LB_{\text{worktime}}}{UB_{\text{worktime}} - LB_{\text{worktime}} + \varepsilon} \quad (14)$$

$$+ \alpha_3 \cdot \frac{1}{J} \sum_{j \in \mathcal{J}} p_g(j) + \alpha_4 \cdot \frac{1}{N} \sum_{n \in \mathcal{N}} p_{\mathcal{N}}(n) \quad (15)$$

$$+\alpha_5 \cdot \frac{1}{N} \sum_{n \in \mathcal{N}} p_{sc}(n) \quad (16)$$

$$\text{and} \quad \alpha_1 + \dots + \alpha_5 = 1, \alpha_i \geq 0 \quad (17)$$

Whereas (15) and (16) only sum up all penalties, and normalize these numbers, (13) and (14) are a slightly more complicated. In (13), the double sum accumulates the travel-times needed to travel between any two consecutive jobs for all schedules. We normalize that value by relating it to some upper and lower bounds for the total travel-time of a given instance. E.g. we can set $LB_{traveltime}$ equal to the sum of the $J - n$ smallest travel-times calculated between jobs, and accordingly, we use the $J - n$ largest travel-times for $UB_{traveltime}$. The ϵ ensures a valid fraction in case the lower and upper bound overlap. Similarly, (14) models the total working time of a nurse. We can set $UB_{worktime} = \sum_{n \in \mathcal{N}} \max_time_{\mathcal{N}}(n)$ and $LB_{worktime} = \sum_{n \in \mathcal{N}} \min_time_{\mathcal{N}}(n)$.

The model defined above can represent several NP-hard optimization problems. E.g. we get a multi-TSP with time windows, even if we ignore qualifications (8), (12), (16), work time limits (5), (6), and soft time windows (11), (15). Similar adaptations lead to multi-processor scheduling or set covering problems (see [10]).

3 Solving Home Health Care Problems

Our model defined above is a hybrid of a rostering model and a routing model. Good approaches were presented for both models in literature, and we will re-use some of the ideas presented previously to build our heuristics. However, it is not possible to use a two-stage approach that first generates feasible routes, and then assigns nurses to them (violation of hard/soft qualification and nurses' time window constraints). Also the vice versa approach — assigning jobs to nurses and then generating routes — will more than likely produce infeasible or disproportional expensive routes. Therefore, only an integrated approach that considers time scheduling, rostering, and route planning simultaneously is appropriated for the HHC problem as a whole.

Our approach interweaves two parts: (a) finding a partition of jobs to nurses, and (b) finding an optimal sequencing for each such partition. The latter one contains in our case a TSP with time windows and is therefore NP-hard. While looking for a valid partition we always evaluate implied tours, and tour quality feeds back to partitioning.

We have different methods to (a): Initial heuristics that quickly generate an initial solution and two improvement heuristics which take a solution and try to improve it via local exchanges. All of these need information regarding a good sequence of jobs within a roster (part (b)), which we determine via a combined CP and LP approach (Sec. 3.2). As a first step, however, we try to reduce the data complexity of a given instance.

3.1 Preprocessing

During the initial data preprocessing step, we determine which job/nurse pair is compatible with the hard qualification constraint (8) and time window constraint (7), and we store this information. Should there exists a job that can be done by only one nurse, we fix this assignment (without fixing the time at which this job has to be done). Also, we compute the soft constraint values that include all preference parameters. Time windows of jobs can then be shrunk to the earliest or latest time at which a nurse is available (7). In the last step we determine precedences implied

w is rated by the global bound on working time. As stated before, for a given order of jobs the starting times found by the LP are optimal with respect to time window penalties and working time.

3.2.2 Generating Valid Orderings

The generation of all feasible orderings for jobs $\{j_1, \dots, j_k\}$ now remains. Theoretically, this involves solving $O(k!)$ many LPs. In our case, very few permutations correspond to feasible orderings, though. Thus the algorithm is very fast with typical input (see Sec. 5.1). This is due to the fact that in HHC many jobs have to be performed in the morning, after lunch and at bedtime. Hence, there is a lot of overlapping at these times. Intensive care, on the other hand, is provided all day, but usually takes much longer time and thus also overlaps with other long running jobs. We calculate all permutations that correspond to feasible orderings of jobs via a recursive function and solve the LP described above.

Algorithm 1 Find best sequence for set U

generateSequence (sequence S , set U)

```

1: // propagate implied precedence constraints
2: for  $i \leftarrow 1 \dots \text{length}(S) + 1$  do
3:   if ( $U = \emptyset$ ) then goto 12
4:   for all  $j \in U$  do
5:     if (precedence for  $j$  is: after  $S_{i-1}$  and before  $S_i$ ) then
6:       if (inserting  $j$  between  $S_{i-1}$  and  $S_i$  does not violate any time window) then
7:          $U \leftarrow U \setminus \{j\}$ 
8:         insert  $j$  between  $S_{i-1}$  and  $S_i$ 
9:         goto 4 // restart loop at first element in  $U$ 
10:      else
11:        return // infeasible
12: // tree traversal
13: if ( $U = \emptyset$ ) then
14:   solve LP (18), and eventually update  $bestSol$ 
15: else
16:   select  $j \in U$ 
17:   for  $i \leftarrow 1 \dots \text{length}(S) + 1$  do
18:     if (inserting  $j$  between  $S_{i-1}$  and  $S_i$  does not violate any time window) then
19:        $S' = S$ ;
20:       insert  $j$  between  $S'_{i-1}$  and  $S'_i$ 
21:       generateSequence ( $S', U \setminus \{j\}$ ) // recursive call

```

Algorithm 1 describes the procedure: Initially called with $S = \emptyset$, $U = \{j_1, \dots, j_k\}$, the algorithm moves an element of U to any feasible position in S , and recursively checks for possible extensions of S (lines 15–20). (To ease the notation assumed that inserting before the first or after the last element of S can also be represented by an insertion between two consecutive elements). If U becomes empty, S is a valid ordering, and (18) provides optimal starting times for that ordering (lines 12,13). After termination, $bestSol$ contains the best ordering found.

We use propagation to fix any job that needs to be included between two other jobs (lines 1–11) and we stop the current recursion as soon as we find a job for which we cannot fulfill the

precedence constraints in the schedule S currently under construction (lines 6 and 10,11). I.e. we have to revise an earlier branching decision, if inserting a job at a required position is not possible because some other time window on the schedule or the nurse's time window will be violated. Basically, such a test requires checking (3), (4), (7) and it can be improved by applying forward and backward push information (see [21]). Additional constraints on the roster's design may be included in this algorithm as well, and they will further reduce the search space.

To avoid repeated calculations for identical job-sets, we use a cache that stores all optimal schedules found. In the experiments, only some thousand different sequences are determined, whereas some million requests are answered by the cache (see Sec. 5.1).

3.3 Initial Solutions via Constraint Programming

Initial heuristics are used in obtaining a first solution quickly. They provide us with the first rosters, and if the dispatcher allows more time for optimization, they serve as a starting point for improvement heuristics. Our first approach was to adapt insertion and scheduling heuristics developed for the vehicle routing with time windows ([21]). Though they are very fast — usually only a split second — the solutions obtained turned out not to be applicable, as in most cases not all jobs could be covered by nurses. Therefore, we decided to follow a CP approach within an incomplete tree search. On typical instances, we usually obtain very good starting solutions within a few seconds. Furthermore, we may trade time vs. quality with such an approach as we can stop at any time after having found the first solution and take the best one produced so far.

3.3.1 Formulation

We use a redundant modeling for the HHC problem. We represent the roster for nurse n by a set $R^{(n)}$, $n = 1 \dots N$ and each job $j \in \mathcal{J}$ by an integer variable i_j . In a solution, the set $R^{(n)}$ contains all jobs assigned to nurse n , and variable i_j is the nurse who has to serve job j . The fact that we have to cover all jobs (1) is implied by this model, since each job is linked to a nurse. Initially,

$$\text{pos}(R^{(n)}) = \{j \mid \text{quali}_j(j) \subseteq \text{quali}_{\mathcal{N}}(n)\}, \quad n \in \mathcal{N}, \quad \text{req}(R^{(n)}) = \emptyset, \quad (19)$$

$$\text{pos}(i_j) = \{n \mid \text{quali}_j(j) \subseteq \text{quali}_{\mathcal{N}}(n)\}, \quad \forall j \in \mathcal{J}, \quad \text{req}(i_j) = \emptyset, \quad (20)$$

which already covers (8).¹ Next, we state that the n rosters must not intersect (2) by a global cardinality constraint [20]. We force consistency between roster and job variables by stating

$$\forall n \in \mathcal{N}, j \in \mathcal{J}: \quad j \in R^{(n)} \iff i_j = n. \quad (21)$$

We can improve this model by redundant information. If we know from preprocessing that two jobs j, j' have to be served in parallel, they cannot be assigned to just one nurse. We add

$$i_j \neq i_{j'} \quad \text{for all } j, j' \in \mathcal{J} \text{ that have to be in performed in parallel.} \quad (22)$$

Using the sequencing cache, we can add a forward checking. Whenever the shifting of job j to the current required set of a nurse would result in an infeasible ordering, we can remove that job from the nurse's domain:

$$\forall n \in \mathcal{N}: \quad \forall j \in \text{pos}(R^{(n)}) \setminus \text{req}(R^{(n)}): \quad \text{req}(R^{(n)}) \cup \{j\} \text{ infeasible} \Rightarrow \text{pos}(R^{(n)}) \leftarrow \text{pos}(R^{(n)}) \setminus \{j\} \quad (23)$$

¹A set variable has a current domain of those values which are still possible candidates for an assignment (noted as $\text{pos}()$), and those that are already fixed or required (noted as $\text{req}()$). Integer variables are regarded as set variables which have $|\text{req}()| \leq 1$.

3.3.2 Branching and Tree Traversal

If domain filtering alone does not provide a solution or failure, we have to branch. Our strategy here is to select a job j for which the possible set $\text{pos}(j)$ is the smallest, and to first choose a nurse assignment which will result in the best overall improvement.

Algorithm 2 Goal 1: Branch on job with smallest domain, and assign best nurse first

- 1: $j \leftarrow$ job with minimal domain, that is not yet fixed.
 - 2: $n \leftarrow$ nurse in $\text{pos}(j)$ with best improvement value
 - 3: left branch: $(i_j = n)$ right branch: $(i_j \neq n)$
-

Having produced two new subproblems, we have to select the next one to process. We use Limited Discrepancy Search (LDS) ([13]) to traverse the search tree.

3.4 Improvement Heuristics

A solution can often be improved by applying local changes. Metaheuristics, like Simulated Annealing (SA) ([16]) and Tabu Search (TS) ([11, 12]) are widely used and have been shown to produce good solutions in a reasonable amount of time.

In our approach, both metaheuristics are based on a simple *1-shift*, i.e. removing a job from one place and inserting it elsewhere. To flexibly the operator we allow an insertion at a different set of jobs as well as an insertion at a stock Γ . Accordingly, we may remove a job from a set of rosters or from the stock. Of course, deleting a job and re-inserting it in the same set again is prohibited. Moving a job to the stock is highly penalized by adding +1 to the objective.

The *costs* of a 1-shift are defined as the gain from deleting job j from a roster or the stock minus the loss resulting from inserting it again into a certain position in a roster or into the stock. Using the stock we can start with $\Gamma = \mathcal{J}$ and apply the metaheuristics to an empty solution, or we can use a solution found earlier and try to improve on that. Notice, that we allow intermediate steps with a non-empty stock (i.e. we relax (1)). Any solution returned by either metaheuristics, however, requires all jobs to be assigned to nurses.

Since we calculate optimal sequences via the approach presented in Sec. 3.2, it suffices to partition \mathcal{J} into good subsets. As with the CP approach, LP (18) can only be applied, if we replace the minimum work time constraint (#) for subsets under construction by $w \geq 0$. (Because of space limitation we will only present the TS module in more detail.)

Tabu Search

The general strategy of Tabu Search is to systematically explore all possible moves from the current to a neighboring solution. The move leading to the best (non-tabu) neighboring solution is accepted, even if this results in a deterioration of the objective function. To prevent the search from cycling, a tabu list which stores the inverse move for a certain number of iterations is used. Thus, all solutions which can be obtained by applying a move stored in the tabu list are not considered. An aspiration criterion allows us to override this rule, if a move improves the best global solution found so far.

We perform the best 1-shift among all possible ones (in the above sense) to obtain a different solution. I.e., we have to find an optimal sequencing for each member of the 1-shift neighborhood. We set the inverse of that move tabu for the next 10 iterations and update a table *fr* counting how often a certain job is assigned to a nurse. *fr* is used for *diversification* by adding

a specific penalty for frequently used moves. By doing so, we gradually manipulate the cost function into moving the search away from some hot spots. The penalty term represents the additional costs incurred by moving job j to nurse n . It was originally proposed by [23]:

$$p(j, n) = \Delta_{\max} \cdot \frac{fr(j, n)}{fr_{\max}},$$

where $fr(j, n)$ is the frequency of moving job j to nurse n , fr_{\max} is the maximal frequency over all nurses and jobs, and Δ_{\max} is the maximal absolute difference between two consecutive moves performed so far (excluding moves from/to the stock because of their high extra penalty).

Diversification penalties are added to the original objective if no global improving solution is found for some iterations (e.g. 2000). As soon as we find an improving solution we switch back to the original objective function. Should we stall for longer time, we terminate the search. We do not consider any specific intensification within the pure TS.

4 A Hybrid Solution Approach

As we will see in the experimental evaluation, the approaches we have just presented are able to quickly find good operational solutions. There are two points, which can be improved further: Firstly, if time allows, we would like to optimize more, and achieve better plans than those found so far. And secondly, we would like to collect several diverse plans: We experienced that the acceptance of computer generated solutions increases if not only one optimized plan is presented, but also some alternative plans, and dispatchers as well as nurses know that the final decision is made by a human.

In this section we discuss a hybridization technique which accounts for both points. The approach offers a diverse variety of very good solutions to the dispatcher and lets him/her decide which roster is best.

4.1 The Solution Pool Concept

Applying only one improving algorithm to only the best initial solution is not likely to produce the best possible solution. This problem has been recognized in several application fields and approaches based on the concept of a solution pool have turned out to be quite powerful in practical experiments (see [22]).

The idea of a solution pool is to store some intermediate solutions generated via the improvement heuristics and exploit the stored “know-how” for designing new solutions. In a TS context, such an approach refers to the utilization of long term memory (see [11]). In our approach, we apply different heuristics to the solutions in the pool and we use statistical measures for good solutions.

4.2 Using a Good Solution to Improve the Search

Let Ω be a set of solutions found by some heuristics. In the beginning Ω contains solutions found by initial heuristics (IH). In the optimization loop, we apply our improvement heuristics to each solution in Ω , and replace the old solution by the improved one (see Alg. 3).

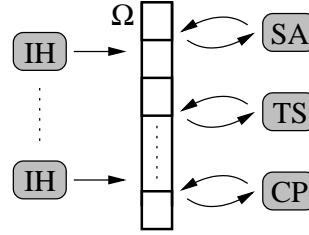
Whereas TS and SA simply start with the solution S provided by the pool, the CP approach has to be adapted slightly. We use the general settings described in Sec. 3.3, and change the branching scheme of goal 1 (Algorithm 2) to consider decisions in S as well. We modify the

Algorithm 3 A pool Ω stores initial and improved solutions found during optimization

```

1:  $P \leftarrow$  problem instance
2:  $\Omega \leftarrow \emptyset$ 
3: while ( $\Omega$  not full) do
4:    $S \leftarrow \text{IH}(P)$ ;  $\Omega \leftarrow \Omega \cup \{S\}$ 
5:   repeat
6:     select  $S \in \Omega$ ;  $\Omega \leftarrow \Omega \setminus \{S\}$ 
7:     select a heuristic method  $h \in \{\text{TS}, \text{CP}, \text{SA}\}$ 
8:      $S' \leftarrow h(S, P)$ ;  $\Omega \leftarrow \Omega \cup \{S'\}$ 
9:   until (termination criterion)
10: return all solutions in  $\Omega$ 

```



search order with respect to soft constraint satisfaction (provided by an input solution). Given a solution S , we can sort the $sc(n, j)$ values of that solution in a preprocessing step. In goal 3 (Alg. 4), we chose j as the first job in this order that is not yet assigned. We try to use the same nurse n assigned to j as in solution S . The ratio behind this is that in a good solution many assignments already correspond to assignments in an optimal solution, and we will use as many of these assignments as possible. If the nurse used in S is not available (e.g. because some propagation or earlier branching has assigned her elsewhere), we select the next nurse as in goal 1. This results in a CP based tree search where valid solutions which are similar to S get built first. The corresponding goal 3 is described in Alg. 4.

Algorithm 4 Goal 3: Branch on job with best soft constraint values first, and assign the nurse that was also used in solution S

preprocessing-step: sort $sc(n, j)$ for all (j, n) pairs found in S

```

1:  $j \leftarrow$  first job in sorted list that is not yet fixed
2:  $n \leftarrow$  nurse that was also used in solution  $S$  for job  $j$  if possible
3: else  $n \leftarrow$  nurse in  $\text{pos}(j)$  with best improvement value (as in goal 1)
4: left branch:  $(i_j = n)$    right branch:  $(i_j \neq n)$ 

```

4.3 Using the Essence of All Solutions

A further solution improvement is gained by using statistical information from the solution pool to guide the CP approach. The idea is to detect job/nurse pairs, that often occur in solutions with high quality, and to consider those assignments early in a systematic search. Vice versa, pairs occurring in low quality solutions only should be considered late in a systematic search. This idea leads to a modified variable ordering and variable assignment goal within the CP approach.

Let $\Omega = \{S_1, \dots, S_k\}$ be a pool of k (diverse) solutions, $v(S)$ be the value of solution S . For any nurse $n = 1, \dots, N$ and any job $j = 1, \dots, J$, let $\mu(n, j)$ be the accumulated objective value of all solutions in Ω assigning job j to nurse n . Accordingly, let $\kappa(n, j)$ be a counter for the number of solutions containing such an assignment. We consider those assignments that have a low value of $\frac{\mu(n, j)}{\kappa(n, j)}$ as “good”. This value is the average quality of solutions containing an assignment of j to n . We order the pairs (n, j) , $n = 1, \dots, N$, $j = 1, \dots, J$ according to increasing values $\frac{\mu(n, j)}{\kappa(n, j)}$. In the branching decision described in Sec. 3.3 we replace the variable selection by a selection step

which takes the first pair (n, j) for which an assignment is possible. Then we assign j to nurse n on the left branch, and we exclude j from the possible set of n on the right branch.

Since we usually only perform an incomplete search, it is likely that assignments made in the first part of the search tree will be included in the best solution found after interrupting the search. To prevent the statistical information gathered in $\mu(n, j)$ and $\kappa(n, j)$ from being part of a self-stabilizing process, we refine the collection process as follows: We increase the value $\frac{\mu(n, j)}{\kappa(n, j)}$ by an additional penalty depending on the frequency of a certain assignment. The more often an assignment is used, the higher the penalty added. We use

$$\frac{\mu(n, j)}{\kappa(n, j)} + \left(\frac{\kappa(n, j)}{\max_{n', j'} \{\kappa(n', j')\}} \cdot [\max_{n', j'} \{\mu(n', j')\} - \min_{n', j'} \{\mu(n', j')\}] \right). \quad (24)$$

In doing so, high quality parts are still assigned first, but rarely used parts are preferred to more frequently used ones. We assume this strategy produces more diverse solutions than the pure quality ordering alone. The algorithmic framework is presented in Alg. 5.

Algorithm 5 Goal 4: Branch on job/nurse pairs that occur in good solutions found earlier

preprocessing-step: sort (n, j) according to increasing values obtained by (24)

- 1: select (n, j) as the first in the previously sorted list that is not yet fixed.
 - 2: if no such pair exists, select (n, j) as in goal 3
 - 3: left branch: $(i_j = n)$ right branch: $(i_j \neq n)$
-

5 Numerical Evaluation

All algorithms were coded in C++ and compiled by the GNU g++ 2.95.3 compiler using full optimization. Our benchmark tests were run on a Pentium III-933 PC with 512MB RAM operating Linux kernel 2.4.19. For solving LP (18) we use ILOG CPLEX 7.5 [14], and for the CP approach we apply ILOG SOLVER 5.2 [15]. SA was implemented using PARSA [17].

We used 10 synthetic test scenarios, containing between 20 and 50 nurses, and between 111 and 326 jobs. The data was generated according to real-world input. Each job lasts between 6 and 72 minutes, nurses' hard time windows between 5 and 9 hours. Locations were chosen randomly, and euclidean distances were calculated between these locations. Also, the soft constraint factor for each job was selected randomly.

The tables and figures presented in the following show the quality value as defined by the objective. It contains routing quality, time window penalties as well as the soft qualification measure. All terms were equally weighted by $\alpha_i = \frac{1}{5}$. The run times are given in seconds. To reflect a real-world setting, all run times are limited to 600sec or 840sec, respectively. A more detailed experimental analysis of our methods is given by [4].

5.1 Optimal Sequences

The table in Fig. 3 shows that enumerating all possible orderings for a given set of jobs does not drastically boost the computing time, if we use time window constraints and job precedences to limit the enumeration tree. Algorithm 1 builds only few sequences and even less LPs have to be solved. Notice, that a complete enumeration for a roster of size k would result in $k!$ recursive calls. E.g. there are 6 227 020 800 possible permutations for a 13-job-roster whereas we need

roster size	recursive calls	LP calls
2	1.094	0.962
3	1.356	1.028
4	1.443	1.015
5	1.665	1.073
6	2.057	1.139
7	2.837	1.344
8	3.986	1.555
9	6.548	2.585
10	7.936	2.988
11	6.566	1.882
12	7.342	1.911
13	18.398	4.162
14	24.560	4.317
15	36.350	3.649

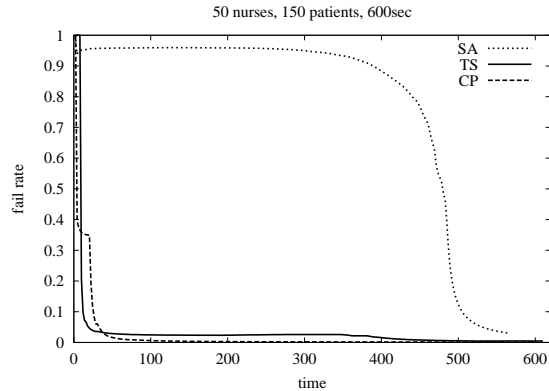


Figure 3: Sequencing and Cache

sizes $ \mathcal{N} - \mathcal{P} - \mathcal{J} $	first solution		after 10sec	sizes $ \mathcal{N} - \mathcal{P} - \mathcal{J} $	first solution		after 10sec
	time in sec.	objective	objective		time in sec.	objective	objective
20-40-111	1.70	0.20128	0.17584	30-120-177	6.02	0.18221	0.17958
20-60-123	2.70	0.22422	0.21821	30-150-195	7.36	0.18859	0.18565
20-80-137	3.49	0.19887	0.18732	50-100-304	15.16	0.18155	—
30-60-184	5.14	0.19603	0.19501	50-150-319	20.19	0.16904	—
30-90-184	6.34	0.19293	0.18849	50-200-326	22.45	0.17142	—

Figure 4: Finding initial solutions via CP

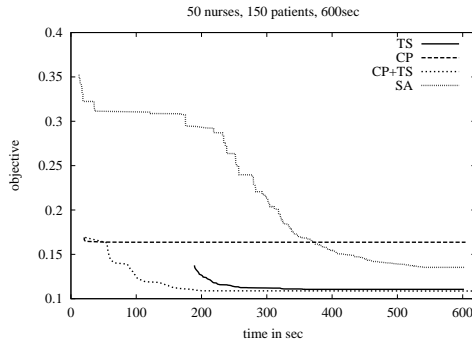
less than 19 calls on the average. An even smaller number corresponds to valid orderings (which have to be processed by the LP). These encouraging results are due to tight time windows which are common to real-world HHCP instances.

In Fig. 3 the efficiency of the sequence cache defined in Sec. 3.2.2 is shown for the basic solution methods. We can see that CP requests many identical rosters due to the systematic search. TS benefits considerably from the cache when enumerating the neighborhood. Due to its randomness SA arbitrarily jumps around in the solution space. SA is forced to behave more locally only after a decrease in temperature and thus can benefit from previously optimized rosters. The small bend of the CP curve is likely due to the implementation of LDS in ILOG SOLVER, see [15].

5.2 Initial Solutions via Constraint Programming

In Fig. 4 we show results obtained for the initial solution. That is, we applied CP (using goal 1) to each benchmark set until the first solution was found. Whenever the first solution was found in less than 10 seconds, we also show the best value found after 10sec. (To get an idea of the quality obtained, these results can be compared to those in Fig. 5).

As can be seen in Fig. 4, a good starting solution can be found in a short time even for larger instances. Typically, the first solution, and the one found after 10sec are of the same quality. The 50 nurse instances take considerably longer than the smaller ones. There are two possible reasons for this: The CP heuristics might be bad, making many wrong decisions before finding the first solution, or the time used in each choice point may be high. When analyzing the results in more detail we found that the number of choice points explored, and the number of failures encountered were reasonable. The time spent for sequencing sets, however dominates the overall running time. In the beginning sequencing rosters is quite expensive in terms of computing



sizes $ \mathcal{A} - \mathcal{P} - \mathcal{J} $	objective after 600sec			
	CP	SA	TS	CP+TS
20-40-111	0.17107	0.14011	0.13019	0.13207
20-60-123	0.20776	0.16964	0.15361	0.15938
20-80-137	0.18580	0.14023	—	0.13108
30-60-184	0.19091	0.15739	—	0.13089
30-90-184	0.18652	0.13759	0.12244	0.11926
30-120-177	0.17680	0.13931	0.12199	0.12113
30-150-195	0.18341	0.14122	0.12139	0.12093
50-100-304	0.17476	0.14311	0.11955	0.11642
50-150-319	0.16364	0.13528	0.11057	0.10869
50-200-326	0.16706	0.13768	—	0.11582

Figure 5: Results for CP, SA, and TS started from an empty solution, and for TS started on the first solution found by CP (CP+TS). A dash indicates that no solution was found. Each approach ran for 600sec.

time, since no information is stored in the cache. Later on, many sequencing requests can be answered by the cache. Thus succeeding computations benefit from the bigger effort required in the beginning.

5.3 Comparing the Heuristics

Unfortunately, if CP continues running for 10min, the good start is not followed by a good convergence. The first column in the table of Fig. 5 gives solution values which are only slightly better than the initial ones found by CP within the first few seconds. SA (started on an empty initial solution) produces better solutions than CP, but cannot compete against TS (also started on an empty solution) as can be seen in column three of that table. TS outdoes both of the previously mentioned approaches — if it can find a solution at all. In three cases TS seems to be too aggressive, and it is not able to generate a feasible solution at all within 600 seconds.

Although SA and TS are both based on the same 1-shift neighborhood, their solution quality differs significantly. Such a behavior seems to be partly approach immanent (see e.g. [19]). Another reason is, again, the sequencing. Whereas TS and CP systematically check their neighborhood and thus can reuse previously generated sequencing information, SA “jumps” randomly, resulting in much more cache fails and thus, much more calculations of sequences (see Fig. 3).

Motivated by the success of TS we started TS on the first solution found by CP (referred to as “CP+TS”). Not only does this approach ensure obtaining a feasible solution, it also appears to be the best approach in this test. Except for the smallest instances, it surpasses all other methods.

Figure 5 shows a typical plot comparing our four approaches. SA when starting from scratch takes quite some time to improve the solution. SA does not reach the solution quality of TS, or CP+TS. CP quickly finds a solution, but has difficulties improving it. TS needs some time to find a valid starting point which it then quickly improves. Combining CP and TS brings together the advantages of both approaches.

5.4 Combining Approaches

Instead of stopping the search after the termination of TS, we can also trade our computing time differently. In one test we ran CP for two minutes to find 10 different initial solutions which we stored in Ω . Then each of these solutions was optimized via a TS limited to one minute each

sizes $ \mathcal{N} , \mathcal{P} , \mathcal{J} $	objective		
	CP+TS	CP+TS loop	10 * CP+TS
20-40-111	0.13207	0.13655	0.13053
20-60-123	0.15938	0.15896	0.15345
20-80-137	0.13108	0.13016	0.12945
30-60-184	0.13089	0.13278	0.12980
30-90-184	0.11926	0.11804	0.11869
30-120-177	0.12113	0.12510	0.12092
30-150-195	0.12093	0.12182	0.12081
50-100-304	0.11642	0.11385	0.11520
50-150-319	0.10869	0.10809	0.10704
50-200-326	0.11582	0.11378	0.11503

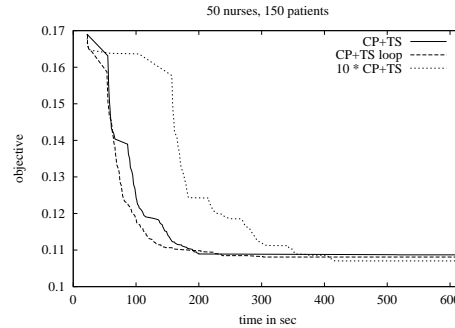


Figure 6: Combined approaches. Results for CP+TS, alternating CP and TS (CP+TS loop), and applying TS to 10 initial CP solutions (10* CP+TS). Running times between 600sec and 840sec.

(starting diversification after 500 non-improving rounds). Finally, all information collected in the $\mu(n, j)$, and $\kappa(n, j)$ tables was used to run CP using goal 4 for additional two minutes.

The other test consists of running CP for an initial solution (goal 1), followed by alternating runs of TS and CP on the best solution found (we used goal 2 which produced a similar solution quality as goal 3). The time limit of a round was raised whenever the previous round did not improve the global best solution.

In Table 6 we present the results obtained by these two methods. Evidently, applying short optimization runs to a larger set of solutions outperforms both, the simple approach (CP+TS) considered in the previous section, as well as the alternating approach. Goal 4 was seldom able to find even better solutions. Figure 6 compares the combined approaches for $|\Omega| \leq 10$.

6 Conclusions

We presented a compact model to the HHC problem which is flexible enough to break down most real-world HHC problems of different characteristics. Furthermore, we developed several solution approaches for the model. The approaches find good quality rosters that satisfy routing, qualification, time windows and soft constraints.

We combined the effectiveness of LP for finding optimal starting points of jobs and the effectiveness of CP for generating feasible ordering. This module is utilized by CP and local search methods. We can offer several good solutions to the end users by using a pool of solutions, and we can use information collected in the pool to improve the quality of these solutions. Experimental results show significant gains when these different methods are combined.

The industrial prototype is currently undergoing intensive user evaluation, where solutions generated by our optimization methods are competing in real-world scenarios. In a second step, requests for additional legal and company constraints, or credit point systems (contributed by researchers from ergonomics) may be integrated into our algorithmic framework.

References

- [1] S. Abdennadher and H. Schlenker. Nurse scheduling using constraint logic programming. In *Proc. of the 11th Conf. on Innovative Applic. of AI*, pages 838–843, Menlo Park, California, 1999. AAAI Press.

- [2] C. Barnhart, Pamela Vance, E. L. Johnson, and George L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 43:188–200, 1997.
- [3] Sachidanand V. Begur, David M. Miller, and Jerry R. Weaver. An integrated spatial dss for scheduling and routing home-health-care nurses. *Interfaces*, 27(4):35–48, 1997.
- [4] Stefan Bertels. Integrierte Personal- und Tourenplanung am Beispiel ambulanter Krankenpflege. Diploma thesis, University of Paderborn, 2002.
- [5] E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *SEAL'98*, volume 1585 of *LNAI*, pages 187–194. Springer, 1999.
- [6] A. Caprara, Filippo Focacci, E. Lamma, P. Mello, Michela Milano, Paolo Toth, and Daniele Vigo. Integrating constraint logic programming and operations research techniques for the crew rostering problem. *Software – Practice and Experience*, 28(1):49–76, 1998.
- [7] B. M. W. Cheng, J. H. M. Lee, and J. C. K. Wu. A nurse rostering system using constraint programming and redundant modeling. *IEEE Trans. in Information Technology in Biomedicine*, 1(1):44–54, 1997.
- [8] Eddie Cheng and Jennifer Lynn Rich. A home health care routing and scheduling problem. Technical Report CAAM TR98-04, Rice University, 1998. (an earlier version was presented at *ISMP'97*).
- [9] Torsten Fahle, Ulrich Junker, S.E. Karisch, Niklas Kohl, Meinolf Sellmann, and Bo Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.
- [10] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.
- [11] Fred Glover. Tabu Search—Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [12] Fred Glover. Tabu Search—Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
- [13] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, pages 607–613. Morgan Kaufmann, 1995.
- [14] ILOG. *Ilog Cplex V7.5 Reference manual and User manual*, 2002.
- [15] ILOG. *Ilog Solver V5.2 Reference manual and User manual*, 2002.
- [16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [17] G. Kliewer, K. Klohs, and S. Tschöke. Parallel simulated annealing library (parsa): User manual. Technical report, University of Paderborn, 1999.
- [18] Andrew J. Mason and Mark C Smith. A nested column generator for solving rostering problems with integer programming. In L. Caccetta, K. L. Teo, P. F. Siew, Y. H. Leung, L. S. Jennings, and V. Rehbock, editors, *Int. Conf. on Optimisation : Techniques and Applications*, pages 827–834, 1998.
- [19] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.
- [20] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of AAAI-96*, pages 209–215, 1996.
- [21] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, March–April 1987.
- [22] É. D. Taillard, L. M. Gambardella, Michel Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal on Operational Research*, 135(1):1–16, 2001.
- [23] Éric D. Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 1997.
- [24] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.