Alcom-FT Technical Report Series
ALCOMFT-TR-03-149

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.

Keywords: Rostering, Vehicle Routing, Metaheuristics, Constraint Programming.

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. [36] and e.g. [3, 8, 12]). To obtain applicable solutions, however, it is crucial to solve the

*This work was partly supported by the German Ministry for Education and Research Bmb+f (PARPAP project 01 HR 9955), and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

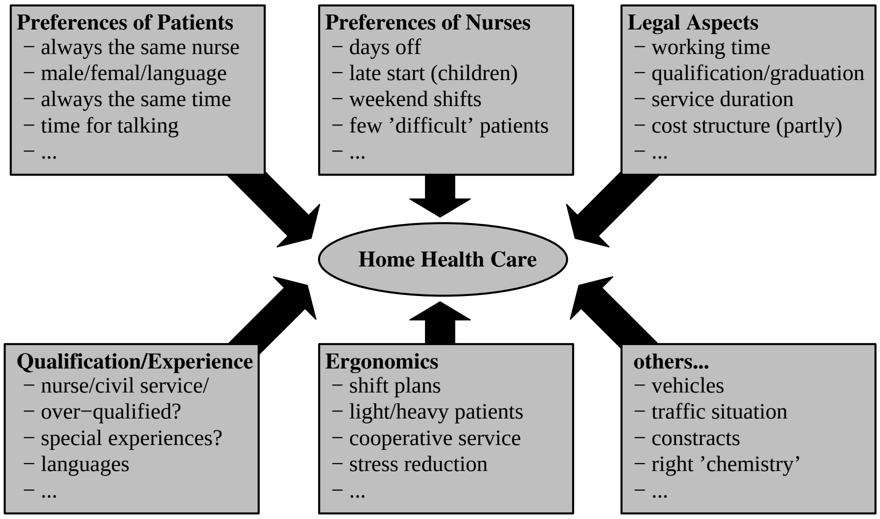


Figure 1: An overview of constraints relevant to the home health care problem.

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. In Figure 1 we present a more detailed view of constraints relevant to the HHC 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. In countries, where health services are partially included in the social system, a company typically gets a fixed amount of money for a given service—independent of the income of the nurse providing the service. Hence, companies prefer schedules where highly qualified staff are only assigned to patients that need that high qualification. Simpler jobs should be assigned to less skilled, and hence less expensive employees.

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.).

1.1 The Industrial Prototype

The project PARPAP (see [5]) brings together end users (home health care companies), universities (computer science and ergonomics) and partners from software industries to develop an optimization and planning tool for the highly constrained problem of staff rostering and routing in the home health care industry. The aim of the project is to model the problem accurately and to develop suitable algorithms for finding good rosters respecting all hard and soft constraints mentioned above. Since rules and regulations for home health care change frequently, and company philosophies differ also, there is a need for flexibility in both, modeling capability and algorithmic approaches.

Apart from the problem's constraints we experienced that people in the health care business keep a critical distance from fully automated decision or planning systems. One idea of PARPAP therefore is to build an optimization system that presents several possible solutions and allows the dispatcher to interact with the system in order to select one plan, or to re-calculate parts of the solution.

Another real-world requirement in the context of PARPAP is runtime limitation. In the application scenario, planners like to have a good solution after at most 10–15 minutes runtime. This obviously 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.

All in all, the software prototype consists of a database and a GUI component, as well as a planning and optimization module. This paper, however, is only dedicated to the optimization modules.

1.2 Literature Review

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

Two related topics, however, have attracted more researchers: Planning systems for hospitals model some aspects that are also needed for home health care. We only mention [2, 7, 9, 26] as an example. Most of these use constraint programming techniques in order to model and solve the nurse rostering problem. Classical vehicle Routing with Time Windows (see e.g. [13, 15]) reflects the mobility aspect of the problem, but ignores any further restriction.

In Section 2 we will describe how we model the home health care 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. In brief we present extensions of that model in Section 2.1.4. 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 Section 4. The next part of the paper presents numerical results of the various methods, and finally we conclude.

2 A Mathematical Model for the Home Health Care Problem

Staff rostering describes the process of assigning staff to tasks. In HHC, the staff consists of employers with various skills, and the "duties" are the services to be provided to the patients. 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 $\mathcal{N} = \{1,\ldots,N\}$, P patients $\mathcal{P} = \{1,\ldots,P\}$, and a set of J jobs $\mathcal{J} = \{1,\ldots,J\}$. 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:

- $\mathrm{hb}_{\mathcal{J}} : \mathcal{J} \to \mathbb{R}$, $\mathrm{he}_{\mathcal{J}} : \mathcal{J} \to \mathbb{R}$, describe start and end, resp. of a job's *hard* time window.

- $\mathrm{sb}_{\mathcal{J}} : \mathcal{J} \to \mathbb{R}$, $\mathrm{se}_{\mathcal{J}} : \mathcal{J} \to \mathbb{R}$, describe start and end, resp. of a job's *soft* time window.

- $\mathrm{d}_{\mathcal{J}} : \mathcal{J} \to \mathbb{R}$ is the duration of the job, i.e. the time needed to complete a job.

- $\mathrm{hb}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$, $\mathrm{he}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$, $\mathrm{sb}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$, $\mathrm{se}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$, represent the start and end of the *hard* and *soft* time window of a nurse.

- $\mathrm{min\_time}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$ and $\mathrm{max\_time}_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}$ give minimal and maximal working time of a nurse.

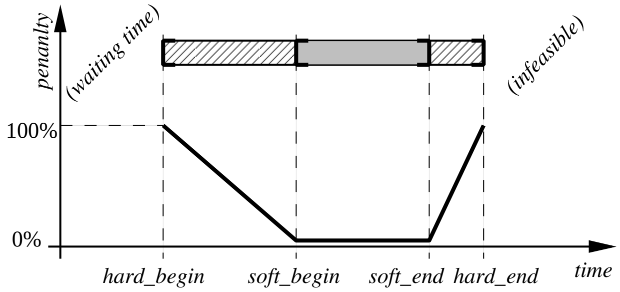


Figure 2: Penalty concept for Time Windows. Arriving before *hard_begin* produces waiting time, arriving after *hard_end* is infeasible. Penalties proportional to earliness or lateness are used for arrivals within the hard, but before or after the soft time window.

Obviously, for a job $j \in \mathcal{J}$ we have $\mathrm{hb}_{\mathcal{J}}(j) \leq \mathrm{sb}_{\mathcal{J}}(j) \leq \mathrm{se}_{\mathcal{J}}(j) \leq \mathrm{he}_{\mathcal{J}}(j)$. The same holds for time windows of nurses. Figure 2 shows an example for these time windows.

Qualifications are the skills possessed by a nurse, or required by a job, respectively. As we will see later, 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:

- $\mathrm{quali}_{\mathcal{J}} : \mathcal{J} \to 2^S$ and $\mathrm{quali}_{\mathcal{N}} : \mathcal{N} \to 2^S$ 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 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:

- $\mathrm{sc} : \mathcal{N} \times \mathcal{J} \to [0 \ldots 1]$.

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:

- $\mathrm{tr\_time} : \mathcal{J} \times \mathcal{J} \to \mathbb{R}$, 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:

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

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

$$\mathrm{hb}_{\mathcal{J}}(j_l^{(i)}) \le t_l^{(i)} \le \mathrm{he}_{\mathcal{J}}(j_l^{(i)}), \quad i = 1\dots N,\ l = 1\dots k_i \tag{3}$$

$$t_l^{(i)} + \mathrm{d}_{\mathcal{J}}(j_l^{(i)}) + \mathrm{tr\_time}(j_l^{(i)}, j_{l+1}^{(i)}) \le t_{l+1}^{(i)}, \quad i = 1\dots N,\ l = 1\dots k_i - 1 \tag{4}$$

$$\mathrm{min\_time}_{\mathcal{N}}(i) \le t_{k_i}^{(i)} + \mathrm{d}_{\mathcal{J}}(j_{k_i}^{(i)}) - t_1^{(i)}, \quad i = 1\dots N \tag{5}$$

$$\mathrm{max\_time}_{\mathcal{N}}(i) \ge t_{k_i}^{(i)} + \mathrm{d}_{\mathcal{J}}(j_{k_i}^{(i)}) - t_1^{(i)}, \quad i = 1\dots N \tag{6}$$

$$\mathrm{hb}_{\mathcal{N}}(i) \le t_1^{(i)} \quad \text{and} \quad t_{k_i}^{(i)} + \mathrm{d}_{\mathcal{J}}(j_{k_i}^{(i)}) \le \mathrm{he}_{\mathcal{N}}(i). \tag{7}$$

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

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 $t_j \in [\mathrm{hb}_{\mathcal{J}}(j), \mathrm{he}_{\mathcal{J}}(j)]$ be the time assigned to job j. Then

$$\mathrm{early}_{\mathcal{J}}(j) := \begin{cases} 0, & \mathrm{sb}_{\mathcal{J}}(j) = \mathrm{hb}_{\mathcal{J}}(j) \\ \frac{\mathrm{sb}_{\mathcal{J}}(j) - t_j}{\mathrm{sb}_{\mathcal{J}}(j) - \mathrm{hb}_{\mathcal{J}}(j)}, & \mathit{else} \end{cases} \tag{9}$$

$$\mathrm{late}_{\mathcal{J}}(j) := \begin{cases} 0, & \mathrm{he}_{\mathcal{J}}(j) = \mathrm{se}_{\mathcal{J}}(j) \\ \frac{t_j - \mathrm{se}_{\mathcal{J}}(j)}{\mathrm{he}_{\mathcal{J}}(j) - \mathrm{se}_{\mathcal{J}}(j)}, & \mathit{else} \end{cases} \tag{10}$$

$$\mathrm{p}_{\mathcal{J}}(j) = \max\{\mathrm{early}_{\mathcal{J}}(j), \mathrm{late}_{\mathcal{J}}(j), 0\} \tag{11}$$

Since t_j is within the soft time window ($\Rightarrow$ $\text{early}_{\mathcal{J}}(j) \leq 0$ and $\text{late}_{\mathcal{J}}(j) \leq 0$), before the soft time window ($\Rightarrow$ $\text{early}_{\mathcal{J}}(j) > 0$ and $\text{late}_{\mathcal{J}}(j) \leq 0$), or after it ($\Rightarrow$ $\text{early}_{\mathcal{J}}(j) \leq 0$ and $\text{late}_{\mathcal{J}}(j) > 0$), choosing $\text{p}_{\mathcal{J}}(j)$ as in (11) gives the correct penalty (see Figure 2).

Violating nurses' soft time window is treated similarly, besides taking care of both, the soft time window's beginning and end. Let k_n denote the last job assigned to nurse n, $n = 1 \dots N$:

$$\begin{aligned}
\text{early}_{\mathcal{N}}(n) &:= \begin{cases} 0, & \text{sb}_{\mathcal{N}}(n) = \text{hb}_{\mathcal{N}}(n) \\ \frac{\text{sb}_{\mathcal{N}}(n) - t_1}{\text{sb}_{\mathcal{N}}(n) - \text{hb}_{\mathcal{N}}(n)}, & else \end{cases} && (12) \\
\text{late}_{\mathcal{N}}(n) &:= \begin{cases} 0, & \text{he}_{\mathcal{N}}(n) = \text{se}_{\mathcal{N}}(n) \\ \frac{t_{k_n} + \text{d}_{\mathcal{J}}(j_{k_n}) - \text{se}_{\mathcal{N}}(n)}{\text{he}_{\mathcal{N}}(n) - \text{se}_{\mathcal{N}}(n)}, & else \end{cases} && (13) \\
\text{p}_{\mathcal{N}}(n) &= \frac{1}{2}\left(\max\{\text{early}_{\mathcal{N}}(n), 0\} + \max\{\text{late}_{\mathcal{N}}(n), 0\}\right) && (14)
\end{aligned}$$

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:

$$\text{p}_{\text{sc}}(n) := \frac{1}{k} \cdot \sum_{l=1}^{k} \text{sc}(n, j_k) \quad n = 1\dots N \tag{15}$$

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:

$$\begin{aligned}
\text{minimize} \quad & \text{obj}(R^{(1)},\dots,R^{(N)}), \qquad \text{where} \\
\text{obj}(R^{(1)},\dots,R^{(N)}) = \; & \alpha_1 \cdot \frac{\sum_{i=1}^{n}\sum_{l=1}^{k_i-1} \text{tr_time}(j_l^{(i)}, j_{l+1}^{(i)}) - LB_{traveltime}}{UB_{traveltime} - LB_{traveltime} + \varepsilon} && (16) \\
& + \alpha_2 \cdot \frac{1}{N} \frac{\sum_{n\in\mathcal{N}} \left[t_{k_n}^{(n)} + \text{d}_{\mathcal{J}}(j_{k_n}^{(n)}) - t_1^{(n)}\right] - LB_{worktime}}{UB_{worktime} - LB_{worktime} + \varepsilon} && (17) \\
& + \alpha_3 \cdot \frac{1}{J} \sum_{j\in\mathcal{J}} \text{p}_{\mathcal{J}}(j) + \alpha_4 \cdot \frac{1}{N} \sum_{n\in\mathcal{N}} \text{p}_{\mathcal{N}}(n) && (18) \\
& + \alpha_5 \cdot \frac{1}{N} \sum_{n\in\mathcal{N}} \text{p}_{\text{sc}}(n) && (19) \\
\text{and} \quad & \alpha_1 + \dots + \alpha_5 = 1, \alpha_i \geq 0 && (20)
\end{aligned}$$

Whereas (18) and (19) only sum up all penalties, and normalize these numbers, (16) and (17) are a slightly more complicated. In (16), 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, (17) models the total working time of a nurse. We can set $UB_{worktime} = \sum_{n \in \mathcal{N}} \text{max\_time}_{\mathcal{N}}(n)$ and $LB_{worktime} = \sum_{n \in \mathcal{N}} \text{min\_time}_{\mathcal{N}}(n)$.

The model defined above can represent several NP-hard optimization problems. E.g. we get a multi-TSP (see [36]) with time windows, even if we ignore qualifications (8), (15), (19), work time limits (5), (6), and soft time windows (11), (18). Similar adaptations lead to multi-processor scheduling or set covering problems (see [14]).

2.1.3 Modeling Different Characteristics

Hard and soft time windows and especially the concept of qualifications allow the modeling of various different situations which occur in typical real-world situations. If patients prefer to be only served by selected nurses, we can model this via soft constraints. A "qualification tag" can be used to distribute staff over several districts. Language requirements can also be modeled by qualifications. Also, different starting and ending locations can be described by introducing dummy jobs that can only be performed by a specific nurse. Thereby we can model starting or ending at the nurse's home, at a patient, or at the care company's office, respectively. Since the preferences of nurses are modeled analogously we have already covered a major part of the constraints in Figure 1.

2.1.4 Extensions of the Model

In the extended model underlying the industrial prototype, we provide some more expressiveness. E.g. for all time windows, there are factors which adjust the penalty term, thus allowing more emphasis on "critical" patients or staff. To account for experience, the time needed to serve a patient depends not only on the job, but also on the assigned nurse. Moreover, we distinguish between several classes of vehicles and use different routes, travel times and distances for these classes. Finally, the objective of the extended model includes additional terms for travel distance.

Some requirements cannot be formulated only by parameters. E.g. by only using parameters we cannot formulate: "No two consecutive jobs should require lifting a heavy patient". However, we can easily deal with such a requirement within our algorithmic framework and the extended model provides rules to efficiently handle these special constraints.[1]

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.

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

[1] Basically, one has to adapt lines 19,20 in Alg. 1. There, we only extend a roster if the structure is feasible according to the additional requirements.

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 by the time windows. That is, for any two jobs we store if they have to be performed in parallel or in an induced order or if they are not related, respectively.

3.2 Sequencing a Roster

Sequencing consists of ordering the jobs and assigning starting times. Due to time window constraints, in the HHC only few permutations correspond to feasible orderings. In our approach we enumerate those orderings by a CP approach, and we use an LP to find optimal start times with respect to the objective.

3.2.1 Optimal Start Times for a Valid Ordering

Given a set $\{j_1, \dots, j_k\}$ and a nurse n we need to find an optimal sequencing (i.e. an ordering of these jobs and an assignment of a start time to each of the jobs) such that the sum of travel times and penalties for not hitting the soft time window is optimal for nurse n. For a given ordering and a given nurse, we can easily calculate penalties by applying (9)–(11). Moreover, a simple LP can be used to optimize the starting times t_l of each job j_l, $l = 1 \dots k$ (see (21)).

We explain LP (21) in more detail: For $l = 1 \dots k$ variable t_l corresponds to the starting time of job l and x_l accounts for soft time window violations at job l. y_1, y_2 is the penalty for violating the nurse's soft time window and w accounts for working time. The first constraint class corresponds to (4), the three lines using x_l refer to (9)–(11), lines 5,6 limit the starting time to the hard time window (3). The nurse's soft time window ((12)–(14)) is respected by lines 7–9 and the corresponding hard time window (7) in lines 13,14. Finally, lines 10–12 correspond to the minimal and maximal work-time constraints (5) and (6), and to work time minimization. If we use the LP in a setting where sets of jobs are generated incrementally we have to replace (#) by $w \geq 0$ to ensure feasibility.

$$
\begin{array}{llrcll}
 & \min & \multicolumn{4}{l}{\frac{\alpha_2}{UB_{worktime}} \cdot w + \frac{\alpha_4}{2\cdot N}\cdot (y_1+y_2) + \frac{\alpha_3}{J}\cdot \sum_{l=1}^{k} x_l} \\
1: & \text{s.t.} & t_{l+1}-t_l & \geq & \mathrm{d}_{\mathcal{J}}(j_l) + \mathrm{tr\_time}(j_l, j_{l+1}) & l=1\ldots k-1 \\
2: & & \mathrm{sb}_{\mathcal{J}}(j_l) - t_l & \leq & x_l \cdot (\mathrm{sb}_{\mathcal{J}}(j_l) - \mathrm{hb}_{\mathcal{J}}(j_l)) & \\
3: & & t_l - \mathrm{se}_{\mathcal{J}}(j_l) & \leq & x_l \cdot (\mathrm{he}_{\mathcal{J}}(j_l) - \mathrm{se}_{\mathcal{J}}(j_l)) & \\
4: & & x_l & \geq & 0 & \Big\} \; l=1\ldots k \\
5: & & t_l & \geq & \mathrm{hb}_{\mathcal{J}}(j_l) & \\
6: & & t_l & \leq & \mathrm{he}_{\mathcal{J}}(j_l) & \\
7: & & \mathrm{sb}_{\mathcal{N}}(n) - t_1 & \leq & y_1 \cdot (\mathrm{sb}_{\mathcal{N}}(n) - \mathrm{hb}_{\mathcal{N}}(n)) & \\
8: & & t_k + \mathrm{d}_{\mathcal{J}}(j_k) - \mathrm{se}_{\mathcal{N}}(n) & \leq & y_2 \cdot (\mathrm{he}_{\mathcal{N}}(n) - \mathrm{se}_{\mathcal{N}}(n)) & \\
9: & & y_1, y_2 & \geq & 0 & \\
10: & & \mathrm{min\_time}_{\mathcal{N}}(n) & \leq & w & (\#) \\
11: & & \mathrm{max\_time}_{\mathcal{N}}(n) & \geq & w & \\
12: & & t_k + \mathrm{d}_{\mathcal{J}}(j_k) - t_1 & = & w & \\
13: & & t_1 & \geq & \mathrm{hb}_{\mathcal{N}}(n) & \\
14: & & t_k + \mathrm{d}_{\mathcal{J}}(j_k) & \leq & \mathrm{he}_{\mathcal{N}}(n) &
\end{array} \tag{21}
$$

In the LP objective we use the weight factors α_2, α_3, and α_4 of the HHC problem. Worktime 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 $\{j_1, \ldots, j_k\}$ now remains. Theoretically, this involves $O(k!)$ permutations. In our case, due to precedence constraints and time windows, very few permutations correspond to feasible orderings, though. This is due to the fact that in home health care 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 describes the procedure: Initially called with $S = \emptyset, U = \{j_1, \ldots, j_k\}$, 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 assume 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 (21) provides optimal starting times for that ordering (lines 12,13). After termination, *bestSol* contains the best ordering found.

Propagation of disjunctive constraints (see [11]) helps 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

Algorithm 1 Find best sequence for set U

generateSequence (sequence S, set U)

```
// try to fix some jobs
// propagate implied precedence constraints
for i := 1...length(S)+1 do
    if (U = ∅)  then goto 12
    for all j ∈ U do
        if (precedence for j is: after S_{i-1} and before S_i) then
            if (inserting j between S_{i-1} and S_i does not violate any time window) then
                U := U \ {j}
                insert j between S_{i-1} and S_i
                goto 4 // restart loop at first element in U
            else
                return // infeasible
// tree traversal
if (U = ∅) then
    solve LP (21), and eventually update bestSol
else
    select j ∈ U
    for i := 1...length(S)+1 do
        if (inserting j between S_{i-1} and S_i does not violate any time window) then
            S' = S;
            insert j between S'_{i-1} and S'_i
            generateSequence (S', U \ {j}) // recursive call
```

violated. Basically, such a test requires checking (3), (4), (7) and it can be improved by applying forward and backward push information (see [33]). 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 Section 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 [33]. 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 constraint programming (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.

For a detailed description of CP we refer to [20, 25, 27, 37] and limit our presentation here to a short reminder. In CP variables $x_1,\dots,x_n$ are represented by their domains $D(x_1),\dots,D(x_n)$. (Arbitrary) constraints $C(x_{i_1},\dots,x_{i_k})$ describe a relation between some variables. The goal is to find an assignment of values to the variables that respects all domains and all constraints.

CPs are usually solved by applying domain reduction techniques. Each variable has a current domain of those values which are still possible candidates for an assignment (noted as pos()), and those that are already fixed or required (noted as req()). Initially, it holds $\text{pos}(x) = D(x)$ and $\text{req}(x) = \emptyset$. There is a domain reduction algorithm associated to each constraint which tries to remove inconsistent values from the domains of its variables. A propagation algorithm propagates these domain changes among the constraints and continues this process until no further constraint can reduce the domains. Within this steady state we may have some empty domains which indicates failure, we may have found a valid assignment corresponding to a solution ($\text{pos}(x) = \text{req}(x)$), or some domains may not be unique, respectively. In the latter case, a tree search algorithm is used to systematically check the search space and domain filtering is applied to every choice point of that tree.

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,

$$\mathrm{pos}(R^{(n)}) = \{j \mid \mathrm{quali}_{\mathcal{J}}(j) \subseteq \mathrm{quali}_{\mathcal{N}}(n)\}, \quad n \in \mathcal{N}, \quad \mathrm{req}(R^{(n)}) = \emptyset, \tag{22}$$

$$\mathrm{pos}(i_j) = \{n \mid \mathrm{quali}_{\mathcal{J}}(j) \subseteq \mathrm{quali}_{\mathcal{N}}(n)\}, \quad \forall\, j \in \mathcal{J}, \quad \mathrm{req}(i_j) = \emptyset, \tag{23}$$

which already covers (8).[2] Next, we state that the n rosters must not intersect (2) by a global cardinality constraint [29]. We force consistency between roster and job variables by stating

$$\forall\, n \in \mathcal{N},\ j \in \mathcal{J}: \quad j \in R^{(n)} \iff i_j = n. \tag{24}$$

We can improve this model by providing 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. Hence, we add

$$i_j \neq i_{j'} \quad \forall j, j' \in \mathcal{J} \text{ that have to be in performed in parallel}. \tag{25}$$

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:

$$\begin{aligned} &\forall\, n \in \mathcal{N}: \\ &\quad \forall j \in \mathrm{pos}(R^{(n)}) \setminus \mathrm{req}(R^{(n)}): \\ &\qquad \mathrm{req}(R^{(n)}) \cup \{j\} \text{ infeasible} \Rightarrow \mathrm{pos}(R^{(n)}) := \mathrm{pos}(R^{(n)}) \setminus \{j\} \end{aligned} \tag{26}$$

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 $\mathrm{pos}(j)$ 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

goal1

j:= job with minimal domain, that is not yet fixed.
n:= nurse in $\mathrm{pos}(j)$ with best improvement value
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) [19] to traverse the search tree. LDS is an iterative search method. In the i-th iteration, it explores all paths from the root node of the tree to a leaf that has i or less discrepancies. The discrepancy of a path is defined as the number of right branches taken in that path. Such a tree traversal is based on the assumption that the heuristics choosing the next variable and assigning the next value are good and only produces few wrong decisions compared to an optimal variable selection. Hence, if we increase i and traverse the tree using LDS, intuitively, we should find a (near) optimal solution quickly.

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

3.4 Improvement Heuristics

A solution can often be improved by applying local changes. Metaheuristics, like Simulated Annealing [23, 38] and Tabu Search [16, 17] 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 Section 3.2, it suffices to partition $\mathcal{J}$ into good subsets. As with the constraint programming approach, LP (21) can only be applied, if we replace the minimum work time constraint (#) for subsets under construction by $w \geq 0$.

3.4.1 Simulated Annealing

Biased by probabilistic choices, Simulated Annealing (SA) tries to randomly pick good moves and to improve the current solution by applying these moves. To escape from local optima, deteriorating moves are accepted with a certain probability which is gradually decreased during the optimization process. Because of its similarity to smelting processes, this probability is called temperature. The algorithm terminates, if the system is regarded as "frozen" – either by a low temperature, or by not moving to a better state.

In our implementation, the starting temperature is determined automatically, using a method of Aarts et al. [1]. We start with some 1-shifts at a low temperature T and increase T until an acceptance rate of 100% is reached. This temperature is then used as the starting point for the cooling scheme.

The next move is determined by choosing a job and a nurse randomly. Improving moves are always accepted, whereas deteriorating moves are only accepted with probability $e^{-\frac{\Delta C}{T}}$, where ΔC is the cost change resulting from the current move (see [23, 38]). After 1000 iterations we decrease the temperature by a factor of 0.9. Should our time limit exceed or the acceptance rate drop below 0.1% we stop. Simulated Annealing is sketched in Algorithm 3.

3.4.2 Tabu Search

The Tabu Search (TS) used follows the ideas proposed by [16, 17]. The general strategy of Tabu Search is to systematically explore all possible moves from the current to

Algorithm 3 Simulated Annealing

SimulatedAnnealing

$T :=$ initial temperature
repeat
  $iterations := 0$
  repeat
    $iterations := iterations + 1$
    select next move randomly
    probe that move; call Algorithm 1 (*sequencing*) for sets changed
    $\Delta C :=$ cost change caused by that move
    if (($\Delta C < 0$) **or** ($e^{-\frac{\Delta C}{T}} >$ rand())) **then**
      perform that move, update current solution
      if (improved globally) **then**
        update best solution;
  until ($iterations = 1000$)
  $T := 0.9 \cdot T$
until (acceptance rate too low) **or** (time() $> limit$)

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 known 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 [34]: $p(j,n) = \Delta_{\max} \cdot \frac{fr(j,n)}{fr_{\max}}$, 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 $\Delta_{\max}$ 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. Algorithm 4 sketches the procedure.

We do not consider any specific intensification within the pure Tabu Search. The hybrid approach described in the next section will introduce such a technique.

Algorithm 4 Tabu Search

TabuSearch

$iterations := 0$
while ($iteration \leq max\_iter$) **and** (time() $\leq limit$) **do**
  if (diversification required) **then**
    select best neighbor using diversification, (non-tabu or global improvement)
  else
    select best neighbor, (non-tabu or global improvement)
  perform move
  $iterations := iterations + 1$; update fr, $tabu$
  if (improved globally) **then**
    update best solution

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.

Before presenting this approach, let us summarize some characteristics of the approaches presented in the previous section:

(i) Different approaches emphasis on different parts of cost- or constraint structure.

(ii) Having found several starting solutions, it is not clear at all which one will be the best starting point for improvement heuristics.

(iii) In many cases improvement heuristics stop because they are trapped in local optima. A different method may easily escape from such an optima.

As a result, 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 been presented. These strategies turn out to be quite powerful in practical experiments (see [35] for examples of different heuristics and different problems).

4.1 The Solution Pool Concept

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 Tabu Search context, such an approach refers to the utilization of long term memory (see [16]). Three different approaches can be identified within such a framework:

Combine parts of old solutions into a new solution: For the VRPTW, Rochard and Taillard [30] collect intermediate solutions of Tabu Search. They periodically pick some of the generated routes at random (giving higher probability to routes from good solutions) and combine them with a new solution. For customers not covered by this method, they add new routes, each containing only one of these customers. The newly generated solution is the starting point for the next Tabu Search optimization. Similar techniques have been used successfully in [31] for the VRPTW, and in [32] for airline crew rostering.

Apply different heuristics to the solutions in the pool: A bunch of heuristics is applied to all solutions stored in the pool. Caused by different characteristics of the heuristics used, solutions trapped in a local optima via one heuristic method can often be improved in turn by subsequent heuristics [6].

Extract statistical data from the pool as a guidance for heuristics: We count how often a certain assignment occurs in the solutions found so far. Then we prefer assignments which occur often to rare ones. Such an approach is very similar to the "fitness" idea used in genetic algorithms (see [18]). In addition, such information is helpful used for diversification as well.

In our approach, we combine two of these ideas, namely applying different heuristics to the solutions in the pool and using statistical measures for good solutions. We do not consider the first approach which produces very good solutions, but needs a considerable amount of running time. Thus it is not feasible in the planning environment we are focusing on, where dispatchers like to see a solution within a few minutes.

4.2 Using a Good Solution to Improve the Search

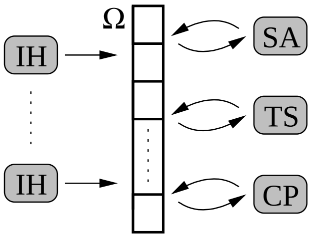
Let Ω be a set of solutions found by some heuristics. In the beginning Ω contains solutions found by initial heuristics (IH). In the optimization loop, we apply our improvement heuristics to each solution in Ω, and replace the old solution by the improved one. (see Algorithm 5).

Whereas Tabu Search and Simulated Annealing simply start with the solution L provided by the pool, the CP approach has to be adapted slightly. We use the general settings described in Section 3.3, and change the branching scheme of goal 1 (Algorithm 2) to consider decisions in L as well. The idea is to select the next job j as before, and to use the same nurse n assigned to j as in solution L. 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 L is not available (e.g. because some propagation or earlier branching has assigned her elsewhere), we select the next nurse as in goal 1. The corresponding goal 2 is described in Algorithm 6.

Furthermore, we try to improve the CP approach by modifying the search order with respect to soft constraint satisfaction (provided by the input solution). Given a solution L, we can sort the $\mathrm{sc}(n, j)$ values of that solution in a preprocessing step. In

Algorithm 5 A pool Ω stores initial and improved solutions found during optimization

$P :=$ problem instance
$\Omega := \emptyset$
while (Ω not full) **do**
  $S := \mathrm{IH}(P)$; $\Omega := \Omega \cup \{S\}$
repeat
  select $S \in \Omega$; $\Omega := \Omega \setminus \{S\}$
  select a heuristic method $h \in \{\text{TS, CP, SA}\}$
  $S' := h(S,P)$; $\Omega := \Omega \cup \{S'\}$
until (termination criterion)
return all solutions in Ω



Algorithm 6 Goal 2: Branch on job with smallest domain and assign the nurse that was also used in solution L

goal2

$j :=$ job with minimal domain, that is not yet fixed
$n :=$ nurse that was also used in solution L for job j if possible
else $n :=$ nurse in $\mathrm{pos}(j)$ with best improvement value (as in goal 1)
left branch: $(i_j = n)$
right branch: $(i_j \neq n)$

goal 3, we chose j as the first job in this order that is not yet assigned, and we determine the nurse as in goal 2. Goal 3 is described as Alg. 7.

Algorithm 7 Goal 3: Branch on job with best soft constraint values first, and assign the nurse that was also used in solution L

preprocessing-step

sort $\mathrm{sc}(n, j)$ for all (j,n) pairs found in L

goal3

$j :=$ first job in sorted list that is not yet fixed
$n :=$ nurse that was also used in solution L for job j if possible
else $n :=$ nurse in $\mathrm{pos}(j)$ with best improvement value (as in goal 1)
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 constraint programming 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 constraint programming 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 Section 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

$$\frac{\mu(n,j)}{\kappa(n,j)} + \left(\frac{\kappa(n,j)}{\max_{n',j'}\{\kappa(n',j')\}} \cdot \left[\max_{n',j'}\{\mu(n',j')\} - \min_{n',j'}\{\mu(n',j')\} \right] \right). \tag{27}$$

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 Algorithm 8.

Algorithm 8 Goal 4: Branch on job/nurse pairs that occur in good solutions found earlier

preprocessing-step

1: sort (n, j) according to increasing values obtained by (27)

goal4

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: $(i_j = n)$
4: 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 (21) we use ILOG CPLEX 7.5 [21], and for the constraint programming approach we apply ILOG SOLVER 5.2 [22]. Simulated Annealing was implemented using PARSA [24].

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.

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

Table 1: Sequencing algorithm efficiency

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.

5.1 Optimal Sequences

Table 1 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 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 for the basic solution methods is shown. 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 peak of the CP+TS curve results from a series of diversification steps that guide the search to a new part of the search space.

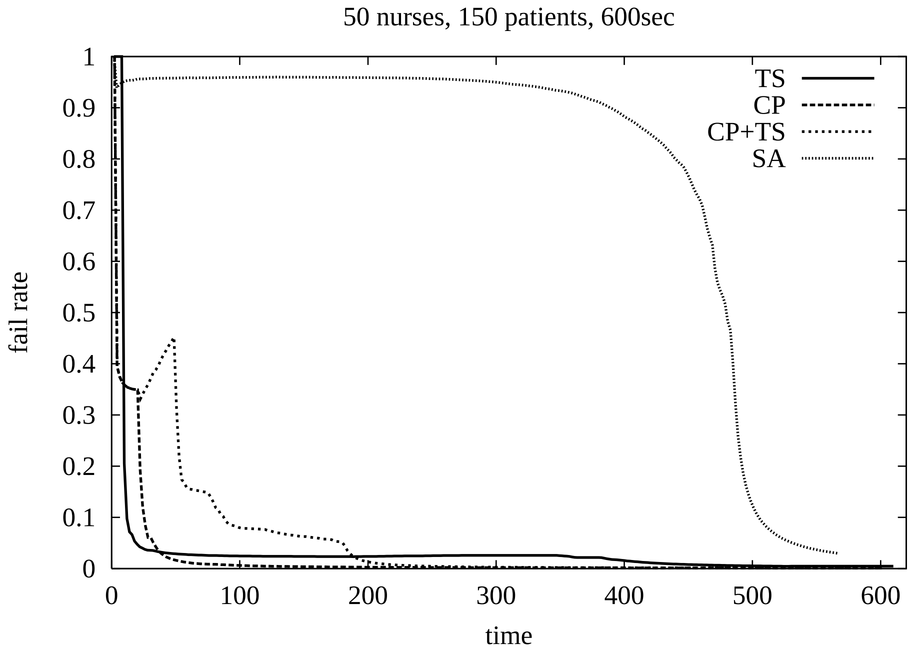


Figure 3: Sequencing cache fail rate

5.2 Initial Solutions via Constraint Programming

In Table 2 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 Table 3).

As can be seen in Table 2, 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 time, since no information is stored in the cache. Later on, many sequencing requests can be answered by the cache. Thus succeeding computations benefit greatly from the bigger effort required in the beginning.

5.3 Comparing the Heuristics

Unfortunately, if CP continues running for 10 min, the good start is not followed by a good convergence. The first column in Table 3 gives solution values which are only slightly better than the initial ones found by CP within the first few seconds. Simulated Annealing (started on an empty initial solution) produces better solutions than CP, but

sizes	first solution		after 10sec
$\lvert\mathcal{N}\rvert$-$\lvert\mathcal{P}\rvert$-$\lvert\mathcal{J}\rvert$	time in sec.	objective	objective
20-40-111	1.70	0.201277	0.175840
20-60-123	2.70	0.224217	0.218209
20-80-137	3.49	0.198873	0.187321
30-60-184	5.14	0.196026	0.195011
30-90-184	6.34	0.192929	0.188494
30-120-177	6.02	0.182207	0.179576
30-150-195	7.36	0.188585	0.185647
50-100-304	15.16	0.181553	—
50-150-319	20.19	0.169044	—
50-200-326	22.45	0.171422	—

Table 2: Results using CP for finding initial solutions

cannot compete against Tabu Search (also started on an empty solution) as can be seen in column three of that table. Tabu Search outdoes both of the previously mentioned approaches — if it can find a solution at all. In three cases Tabu Search seems to be too aggressive, and it is not able to generate a feasible solution at all within 600 seconds.

Interestingly, although Simulated Annealing and Tabu Search 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. [28]). 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 4 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 Tabu Search limited to one minute each (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.

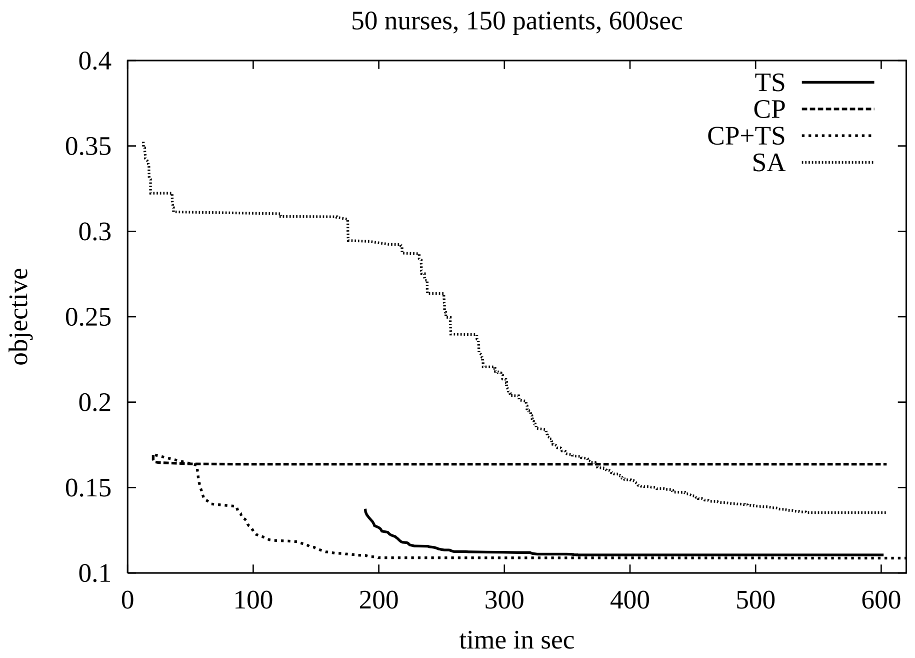


Figure 4: Comparison of the heuristics

sizes	objective after 600sec			
$\|\mathcal{N}\|$-$\|\mathcal{P}\|$-$\|\mathcal{J}\|$	CP	SA	TS	CP+TS
20-40-111	0.171069	0.140114	**0.130188**	0.132068
20-60-123	0.207763	0.169643	**0.153611**	0.159384
20-80-137	0.185795	0.140229	—	**0.131078**
30-60-184	0.190910	0.157389	—	**0.130886**
30-90-184	0.186522	0.137593	0.122436	**0.119256**
30-120-177	0.176804	0.139311	0.121989	**0.121133**
30-150-195	0.183405	0.141220	0.121386	**0.120928**
50-100-304	0.174755	0.143110	0.119550	**0.116424**
50-150-319	0.163636	0.135278	0.110572	**0.108694**
50-200-326	0.167059	0.137677	—	**0.115819**

Table 3: Results for constraint programming (CP), Simulated Annealing (SA), and Tabu Search (TS) started from an empty solution, and for Tabu Search started on the first solution found by constraint programming (CP+TS). A dash indicates that no solution was found. Each approach ran for 600sec.

sizes		objective	
$\lvert\mathcal{N}\rvert$-$\lvert\mathcal{P}\rvert$-$\lvert\mathcal{J}\rvert$	CP+TS	CP+TS loop	10 * CP+TS
20-40-111	0.132068	0.136548	**0.130534**
20-60-123	0.159384	0.158962	**0.153450**
20-80-137	0.131078	0.130157	**0.129449**
30-60-184	0.130886	0.132782	**0.129801**
30-90-184	0.119256	**0.118036**	0.118690
30-120-177	0.121133	0.125103	**0.120917**
30-150-195	0.120928	0.121823	**0.120806**
50-100-304	0.116424	**0.113853**	0.115200
50-150-319	0.108694	0.108091	**0.107043**
50-200-326	0.115819	**0.113778**	0.115029

Table 4: Comparing 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.

In Table 4 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 5 compares the CP+TS method vs. alternating CP and TS vs. TS alone on 10 solutions.

6 Conclusion and Future Work

We presented a compact model to the home health care 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.

Acknowledgment

We would like to thank Eduard Wiebe for his support during the experimentations.

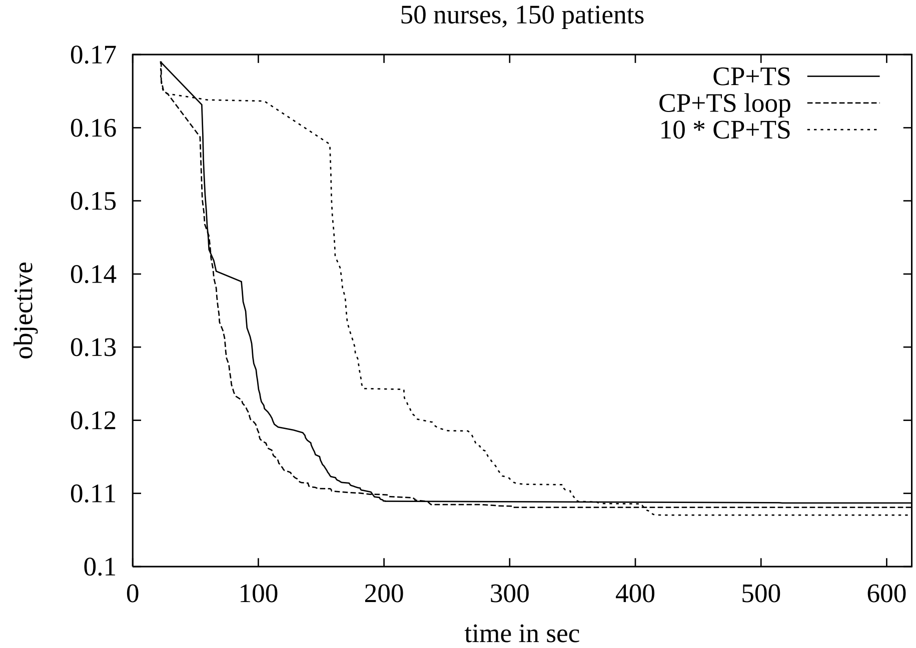


Figure 5: Comparison of combined heuristics

References

[1] E. H. L. Aarts, F. M. J. de Bont, E. H. A. Habers, and P. J. M. van Laarhoven. Parallel implementation of the statistical cooling algorithm. *INTEGRATION, the VLSI Journal*, 4(3):209–238, 1986.

[2] S. Abdennadher and H. Schlenker. Nurse scheduling using constraint logic programming. In *Procs. of the 11th Conf. on Innovative Applications of Artificial Intelligence*, pages 838–843, Menlo Park, California, 1999. AAAI Press.

[3] C. Barnhart, P. Vance, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 43: 188–200, 1997.

[4] S. V. Begur, D. M. Miller, and J. R. Weaver. An integrated spatial dss for scheduling and routing home-health-care nurses. *Interfaces*, 27(4):35–48, 1997.

[5] S. Bertels, T. Fahle, S. Hokemeier, H. Klomp, P. Knauth, M. Rott, M. Templin Partizipative Personaleinsatzplanung für den Ambulanten Pflegedienst – Abschlussbericht. (in german). Eul-Verlag, 2003.

[6] A. L. Bouthillier, T. Crainic, and R. K. Keller. Parallel co-operative evolutionary algorithms for vehicle routing problems with time windows. Talk at PAREO 2000, Paderborn/Germany, 2000.

[7] E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid Tabu Search algorithm for the nurse rostering problem. In *Simulated Evolution And Learning*, volume 1585 of *LNAI*, pages 187–194. Springer, 1999.

[8] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth, and D. Vigo. Integrating constraint logic programming and operations research techniques for the crew rostering problem. *Software – Practice and Experience*, 28(1):49–76, 1998.

[9] B. M. W. Cheng, J. H. M. Lee, and J. C. K. Wu. A nurse rostering system using constraint programming and redundant modeling. *IEEE Transactions in Information Technology in Biomedicine*, 1(1):44–54, 1997.

[10] E. Cheng and J. L. 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*)

[11] U. Dorndorf, T. Phan Huy, and E. Pesch. A Survey of Interval Capacity Consistency Tests for Time- and Resource-Constrained Scheduling. In J. Weglarz (ed.) *Project Scheduling — Recent Models, Algorithms and Applications*, volume 14 of *Internat. Series in Operations Research and Management Science*, chapter 10, Kluwer Academic Publishers, 1998.

[12] T. Fahle, U. Junker, S. E. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.

[13] M. L. Fisher. Vehicle routing. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (editors), *Network Routing*, volume 8 of *Handbooks on Operations Research and Management Science*, pages 1–33. North-Holland, Amsterdam, 1995.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[15] M. Gendreau, G. Laporte, and J.-Y. Potvin. Vehicle routing: modern heuristics. In E. H. L. Aarts and J. K. Lenstra (editors), *Local Search in Combinatorial Optimization*, pages 311–336. John Wiley & Sons, 1997.

[16] F. Glover. Tabu Search—Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[17] F. Glover. Tabu Search—Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[18] Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1984.

[19] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, pages 607–613. Morgan Kaufmann, 1995.

[20] P. Van Hentenryck, Y. Deville, and M. C. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[21] *Ilog Cplex V7.5 Reference manual and User manual*. ILOG, 2002.

[22] *Ilog Solver V5.2 Reference manual and User manual*. ILOG, 2002.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[24] G. Kliewer, K. Klohs, and S. Tschöke. Parallel Simulated Annealing library (parSA): User manual. Technical report, University of Paderborn, 1999.

[25] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 (1):99–118, 1977.

[26] A. J. Mason and M. 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.

[27] U. Montanari. Networks of constraints: fundamental properties and applications. *Information Science*, 7(2):95–132, 1974.

[28] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[29] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of AAAI-96*, pages 209–215, 1996.

[30] Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

[31] J. Schulze and T. Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999.

[32] M. Sellmann, K. Zervoudakis, P. Stamatopoulos, and T. Fahle. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115:207–226, 2002.

[33] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, March–April 1987.

[34] É. D. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A Tabu Search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 1997.

[35] É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal on Operational Research*, 135(1):1–16, 2001.

[36] P. Toth and D. Vigo (editors). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[37] E. P. K. Tsang. *Foundations of Constraint Programming*. Academic Press, 1993.

[38] P. J. M. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Application*. D. Reidel Publishing Company, 1987.