A Hybrid Heuristic for Multimodal Homecare Scheduling

Andrea Rendl[1], Matthias Prandtstetter[1] Gerhard Hiermann[2], Jakob Puchinger[1], and Günther Raidl[2]

[1] AIT Austrian Institute of Technology
Mobility Department, Dynamic Transportation Systems
[2] Vienna University of Technology, Austria

Abstract. In this work we consider solving a large-scale real-world multimodal homecare scheduling problem (MHS), where the objective is to find a roster for homecare nurses that travel from patient to patient in a tour, using different modes of transport, respecting a set of side constraints, and maximising customer, nurse and employer satisfaction. We tackle the problem using a metaheuristical approach, where in a first step, we generate a valid initial solution and in a second step we use different metaheuristics to improve the solution. The initial solution generation is particularly challenging since (1) the instances are huge, (2) many side constraints need to be considered and (3) we may only invest little time. Therefore, we employ a Constraint Programming (CP) -based approach to generate initial solutions. First, we present a novel and efficient constraint model for the MHS. Second, we introduce different clustering techniques to decompose the problem into simpler subproblems. Third, our experimental results show how the metaheuristics' performance is considerably improved by using valid initial solutions and produce useful rosters for the MHS Problem.

1 Introduction

The demand for care of the elderly is growing due to a constant age increase of the population, in particular in today's western world. Thus, efficient health care services are of great significance. In home health care, patients are nursed at home instead of at retirement homes: patients can book different kinds of jobs (e.g. cleaning, cooking, medical services) and nurses of adequate qualification visit patients in a tour. Nurses use different modes of transport, mainly car and public transport, to travel from patient to patient. The objective of the Multimodal Homecare Scheduling problem (MHS) is hence to find a nurse roster where all jobs are assigned to adequate nurses wrt their transport mode, and employer, nurse and customer satisfaction is maximized.

Solving the MHS, i.e. finding an optimal roster, is challenging, since the problem is a combination of two NP-hard problems: vehicle routing with time windows (VRPTW) [4, 5] and nurse rostering (NRP) [7]. Furthermore, since we consider a *large-scale real-world* setup from a Viennese public health care

company, the instances are particularly large (especially compared to instances from the literature, e.g. [18]). We therefore apply a heuristic approach since exact methods are typically too expensive for large instances: in a first step, we generate an initial valid solution that in a second step is improved using a metaheuristic.

Our goal is to construct a general, flexible framework for solving multimodal homecare scheduling problems from different healthcare companies: while the main problem components are the same in every healthcare company (rostering and tour planning), the side constraints, concerning for instance contractual issues or service regulations, typically vary substantially. Therefore, we build a framework with a *flexible architecture*, where side constraints can be easily interchanged.

In this paper we mainly focus on the initial solution generation, one of the main difficulties: first, the problem has many side constraints and is excessively large (5 times larger than typical instances from the litterature), therefore greedy or random construction heuristics have difficulties in finding valid solutions. Second, since we aim at constructing a flexible framework, we want to use an approach that can be easily altered and is not tailored to a specific problem setup. Therefore, we choose Constraint Programming (CP) [19] as main technique to generate initial solutions. CP has a successful history of solving decision problems (i.e. finding initial solutions) [10, 19], and constraints can be easily interchanged which provides the necessary flexibility for the general framework.

We present our CP model and search strategies and give an overview of various clustering approches that we have considered to decompose the problem. Furthermore, we show how valid initial solutions can improve the performance of the metaheuristics, yielding promising rosters for homecare companies.

This paper is structured as follows. First, we specify the Multimodal Homecare Scheduling problem in Section 2 and outline our general solving approach in Section 3. Then we proceed with a detailled discussion of our constraint model for the MHS in Section 4 and introduce problem clustering techniques in Section 5. Finally, we present the results of our empirical evaluation in Section 6 and conclude in Section 7.

2 The Multimodal Homecare Problem

The Multimodal Homecare Scheduling (MHS) problem is concerned with finding a roster for nurses who perform homecare services at patients' homes. Therefore, the problem consists of two components that represent NP-hard problems: (1) a *rostering* component (assigning nurses to jobs, w.r.t. various shift/working hour restrictions) and (2) a *vehicle routing* component (nurses travel in tours from patient to patient, starting at their home). We also integrate a *multimodal* aspect into the problem, since each nurse chooses a specific mode of transport for traveling.

Homecare Jobs. A set of customers $\mathcal{C} = \{1, \dots, C\}$ book one or more jobs, daywise summarized in the set of jobs $\mathcal{J} = \{1, \dots, J\}$. Each job $j \in \mathcal{J}$ has

an associated location loc_j^J (given as GPS coordinates), a start time window tw_j in which the job should start, and a duration dur_j. Furthermore, each job j requires a minimal qualification $q_j^J \in \mathcal{Q}$ by whoever performs the job (e.g. taking blood samples requires a medical qualification). In our problem setup, the set of qualifications $\mathcal{Q}$ is ordered starting with the lowest qualification: $\mathcal{Q} = \{zivi, bd, hh, ph, dgkp\}$. These qualifications represent the different kinds of nurses qualification-wise: community service worker (*zivi*), visiting nurse (*bd*), homecare nurse (*hh*), advanced homecare nurse (*ph*) and medical nurse (*dkgs*).

Nurses. Each job has to be assigned to one of the N nurses given by set $\mathcal{N} = \{1, \ldots, N\}$. Since nurses travel to patients starting from their home, we store the home location loc_i^N of each nurse $i \in \mathcal{N}$ as GPS coordinates. Moreover, each nurse i has a qualification $q_i^N \in \mathcal{Q}$ and may only perform jobs that require a qualification that is less or equal the nurse's maximal qualification (i.e. a medical nurse may perform a lower-qualified homecare job, but a homecare nurse may not perform a medical job). Nurses can only work within specified time windows, given in the set of time windows TW_i for nurse i, that vary from day to day. Furthermore, there exists a general minimal and maximal number of working hours, h_{min} and h_{max}, per day.

Pre-allocated Jobs. In addition to nursing jobs, we also consider *pre-allocated jobs* that nurses perform in addition to nursing (e.g. team meetings, administrative work). These jobs are assigned to a fixed location and a fixed nurse and make up about 5% of all jobs.

Negative Preferences. Patients and nurses may state *negative preferences* against each other; for instance, a nurse might refuse to work for a patient who owns a dog if she has a dog allergy, or a female patient might refuse a male nurse. In such cases, the nurse may not be assigned to the respective job in the roster. We summarize the reasons for refusing a nurse/customer in the set $\mathcal{R} = \{dog, cat, smoker, male, female\}$. Each customer gives 'customer aspects' based on the set of refusal reasons: $\mathcal{A}_k^C = \{a \mid a \in \mathcal{R} \text{ is an aspect for customer } k \in \mathcal{C}\}$. For instance, if customer k has the aspect set $\mathcal{A}_k^C = \{cat, male\}$, then the customer has a cat and does not want to be treated by a male nurse. On the other hand, nurses specify their aspects $\mathcal{A}_i^N = \{a \mid a \in \mathcal{R} \text{ is an aspect for nurse } i \in \mathcal{N}\}$. If nurse i has the aspect set $\mathcal{A}_i^N = \{cat, male\}$, then the nurse has a cat allergy and is male. Therefore, customers and nurses who share one or more aspects may not be assigned to another.

Multimodality. A novel aspect in this work is the consideration of *multimodality*: each nurse i states her preferred mode of transport mot_i where $mot_i \in \{car, pt\}$ (*pt* represents *public transport*), that we consider in creating the roster. To obtain meaningful results, it is crucial to have accurate travel time estimates for each transport mode. Therefore, we obtain travel times according to the GPS coordinates of nurses loc_i^N and jobs loc_j^J based on data from (1) the Viennese public transport system, for public transport and (2) a large set of historical data from Viennese floating car data for transport by car.

2.1 Objective Function

When computing MHS rosters the overall goal is to find valid solutions, i.e. solutions satisfying all hard constraints like legal issues (e.g. maximum working times). Among all valid solutions we want to select the one which is closest to optimality for employer, employees and customers and therefore

1. minimizes the **costs for the employer** (e.g. minimizes travel and working times),
2. maximizes **customer satisfaction** (e.g. by minimizing deviations from customer preferences, such as time window violations)
3. maximizes **nurse satisfaction** (e.g. minimizes deviations from contractual constraints and other nurse preferences)

Therefore, we designed an objective function *ob* which assigns each solution a real number such that valid solutions evaluate to values between 0 and 1, whereas invalid solutions evaluate to values greater than 1. For this purpose, we build a weighted sum of all influencing quantities, like soft constraint violations. Hard constraint violations are respected by increasing the objective value by 1 for each violation. As a side-effect, it is easy to determine how many (hard) constraint violations occurred for a given solution. The following determinants are **hard constraints** and lead therefore to invalid solution when not properly satisfied:

1. All jobs have to be assigned to a well-qualified nurse. Either a missing assignment of a job to a nurse or an insufficient qualification of a nurse are both a hard constraint violation on its own.
2. Availabilities of nurses have to be regarded, i.e. jobs can only be assigned to nurses whenever they are available.
3. Pre-allocated jobs must not be shifted to other times or nurses.

On the other hand, the following requirements are **soft constraints**, whose violation is tolerated but penalized in the objective function:

1. Each customer states a time window in which the job shall be started. Violations of these time windows are penalized using a quadratic function. However, deviations of three hours and above are considered equally bad.
2. Each customer states a desired start time for each job, i.e. a concrete point in time at which the job should be started. Deviations from the start time are linearly penalized, where, similar to time window violations, deviations of one hour and above are assumed to be equally bad.
3. Preferences stated by nurses and customers are handled as soft constraints, since all jobs have to be accomplished (even if preferences are violated).
4. Working times should not exceed the daily *maximal working time* since all additional hours of work are counted as overtime and therefore higher paid (resulting in higher costs for employer).
5. Travel times should be kept as small as possible. On the one hand, journeys between jobs are directly counted as working time (see above). On the other hand, journeys between nurse homes and the first/last job on a day shall be kept as short as possible since they are not paid and therefore lead to (un)satisfied nurses. It is therefore crucial to correctly estimate travel times.

2.2 Related Work

Bertels and Fahle [3] use a combination of *Linear Programming*, *Constraint Programming* for initial solution construction and *Simulated Annealing* and *Tabu Search* as metaheuristics. They consider problem instances for a single day with 20 to 50 nurses and from 111 to 326 jobs and focus on providing reasonable solutions in relatively short time (600 to 840 seconds per instance). The main difference to our problem formulation is that we additionally consider soft constraints (only preferred time windows are considered) and multimodality.

Eveborn et al. [9] use a *set-partitioning* formulation to provide a flexible architecture and solve the problem using *repeated matching*. Starting from an initial matching, this approach iteratively creates a new perfect matching by local improvements and applies splitting until a predefined termination condition is reached. Their problem definition contains many constraints of our work and also considers multimodality. However, the instances they use are rather small (up to 21 nurses and 123 jobs). Based on their results, Eveborn et al. reported that the travelling time savings of their approach are about 20% (on a moderate guess) compared to solutions made by the human counterpart.

Steeg and Schröder [20] use CP to construct an initial solution that is improved by a combination of *Adaptive Large Neighborhood Search* (ALNS) and *Simulated Annealing*. They consider the periodic variant of the problem.

Rasmussen et al. [17] formulate the problem as a *set-partitioning* problem and describe a *branch-and-price* approach. Their model incorporate connected visits (dependencies on the order of the tour) but allows to left some jobs to be unassigned using a priority system. Their tests use real-world instances with up to 15 nurses and 150 jobs.

3 Solving Approach

Our solving approach consists of two main components: (1) an initial solution generation step and (2) an improvement step, where different metaheuristics are applied to systematically enhance the initial solution.

3.1 Initial Solution Generation

First, we determine a *valid* initial solution, i.e. a solution where all jobs are assigned and none of side constraints presented in Section 2 (where we consider all soft constraints as hard constraints) are violated. This is particularly challenging for three reasons: first, we cannot apply a simple construction heuristic, such as a random job-nurse assignment, since many side constraints have to be considered. Second, the instances are large (about 700 jobs/500 available nurses). And third, not too much time should be invested into the initial solution generation, since this step is just the first of several steps in the whole optimization approach.

Constraint Programming (CP) is a particularly attractive candidate for initial solution generation: first, CP is strong in solving discrete decision problems (i.e. finding *first* solutions) [10, 19]. Second, CP search strategies—if set up

correctly—can yield fairly good initial solutions (e.g. favoring low values for job time variables during search produces tight schedules since job times are set as early as possible). Third, the CP model can be easily altered to similar problem setups (of other health care companies) by simply editing side constraints, and hence satisfies our requirement of a flexible system. We further outline our CP-based approach in Section 4 and 5.

As an alternative approach to the CP-based construction heuristic, we employ a simple solution generation heuristic that produces a random nurse-job assignment. The generated solution only guarantees that all jobs are executed by a nurse of adequate qualification and that all pre-allocated jobs are assigned to the right nurse. No side constraints, such as time window restrictions are considered. We use this technique as a backup to the CP approach and to evaluate the influence of the initial solution during the later improvement phase.

3.2 Metaheuristical Optimization

The initial (valid) solutions we receive from the construction heuristics are typically of rather low quality as some aspects of the objective function (such as short travel times) are not considered. Consequently, those solutions cannot be used for real-world rosters. We therefore improve initial solutions using different metaheuristics that we briefly present in this section.

Variable Neighborhood Descent (VND) [12] tries to systematically examine a predefined set of neighborhood structures N_i, with $1 \leq i \leq l_{max}$ where l_{max} denotes the number of defined structures. To speed up the search, the neighborhood structures are ordered such that potentially smaller neighborhoods for a given solution are examined first. As soon as a local optimum is reached (within the i-th neighborhood of solution x), the search is continued with neighborhood $N_{i+1}(x)$. If, however, an improvement is found in the current neighborhood, search is continued with neighborhood structure N_1 again.

We define three neighborhood structures implicitly via three different move types: (1) *shift job* moves one job from one tour to another tour by searching the best matching position in that other tour, (2) *reposition job* repositions a job to the best matching position within its tour (without reordering the other jobs) and (3) *swap nurses* swaps the tours of two nurses. We apply 'first improvement' as step function such that improvements can be gathered rather fast. Although the initial neighborhood order (shift jobs, swap nurses, reposition job) leads to rapid improvements during the starting phase of VND, preliminary tests revealed that dynamic neighborhood reordering [16] leads to more promising results.

General Variable Neighborhood Search (VNS) Since search might get stuck in local optima, VND is typically embedded as local search phase in a general *variable neighborhood search* scheme [12]. Here, the basic idea is to provide diversification in order to escape from local optima by applying random moves in a set of larger neighborhoods (referred to as *shaking*). To broaden search,

the randomness introduced during shaking is enlarged each time the local search phase (e.g. VND) did not improve the current best solution. In our case, shaking is performed by applying $i+1$ random shift job moves within the i-th consecutive iteration of VNS without improvement.

Simulated Annealing Hyper-Heuristic (SAHH) utilizes Simulated Annealing [15] as master, and other local search techniques (*low-level heuristics*) as slaves. Each low-level heuristic has a probability to be selected for the current iteration. These probabilities are changed during the search using an adaptive learning approach, where the selection probability depends on an accept/tested ratio in a predefined learning period.

Our approach is based on Bai et al.'s SAHH algorithm [2] for the Nurse Rostering Problem (NRP), where we replace the low-level heuristics by six low-level heuristics: we apply (1) a classical next improvement strategy and a (2) random-improvement strategy that are adapted according to three the step functions and move types (those described in the previous subsection on the VND), thus, yielding a total number of six low-level heuristics.

Scatter Search uses a small population of diverse solutions and creates subsets of small size from this pool. These subsets are then combined using a solution combination method to create new solutions which are then improved using a local search algorithm. This metaheuristic has shown good results for similar problems, such as the NRP [6].

The initial population is based on the solution retrieved from the CP-based or random construction heuristic. We generate subsets using a classical subset generation approach [11]. For the combination of solutions for each subset we applied a path-relinking algorithm similar to [11] after some preliminary tests with a 'construction by voting' [6] did not deliver promising results on our problem.

4 A Constraint Model for the MHS

In this section we present our CP model for the MHS. Note, that we simplify the problem by solving each instance qualification-wise, since the instances are far too large to be solved within a single CP model. Thus, we create a subinstance I_q from instance I for each qualification $q \in \mathcal{Q}$, such that only jobs requiring qualification q and adequately qualified nurses are contained in I_q. We first start solving the instance for the highest qualification, "medical nursing", that includes all medical nurses and all jobs that require a medical nurse. Then we continue with the second-highest qualification, and so on. Note, that in case we cannot solve a subinstance of qualification q, we iteratively add not yet used nurses of higher qualification to the instance.

We formulate the MHS problem as an extension of the Vehicle Routing Problem with Time Windows (VRPTW) model from [19] and give further details in the following subsections.

Parameter	Description
N	number of nurses
M_u	number of unfixed jobs
M_f	number of fixed jobs
$M = M_u + M_f$	number of jobs
$\mathcal{N} = \{0..M-1\}$	nurses
$\mathcal{J}_u = \{0, .., M_u-1\}$	unfixed jobs
$\mathcal{J}_f = \{M_u, .., M_f-1\}$	fixed jobs
$\mathcal{J} = \{0..M-1\}$	jobs
$\mathcal{V} = \{0..M+2N-1\}$	visits/trips
$\mathcal{T} = \{0..T\}$	timeintervals per day
$\mathcal{Q} = \{zivi, bd, hh, ph, dgkp\}$,	qualifications
$\mathcal{R} = \{dog, cat, smoker, male, female\}$	refusal reasons
$\mathcal{A}_k^J = \{a \mid a \in \mathcal{R}$ is aspect for job $j \in \mathcal{J}\}$	aspects of jobs j
$\mathcal{A}_i^N = \{a \mid a \in \mathcal{R}$ is aspect for nurse $i \in \mathcal{N}\}$.	aspects of nurse i
$\mathcal{S} = \{M..M+N-1\}$	nurse leaves home
$\mathcal{E} = \{M+N..M+2N-1\}$	nurse comes home
$\mathcal{P} = \{car, pt\}$	transport modes
$tw_i^N : \mathcal{N} \to \mathcal{T}^2$	timeslots of nurse $i \in \mathcal{N}$
$tw_j^J : \mathcal{J} \to \mathcal{T}^2$	timeslots of job $j \in \mathcal{J}$
$dur_j : \mathcal{J} \to \mathcal{T}$	duration of job j
$q_i^N : \mathcal{N} \to \mathcal{Q}$	max. qualification of nurse i
$q_j^M : \mathcal{J} \to \mathcal{Q}$	min. qualification for job j
$fixt_j : \mathcal{J}_f \to \mathcal{T}$	time of fixed job j
$fixn_j : \mathcal{J}_f \to \mathcal{N}$	nurse of fixed job j
$tt_{ij} : \mathcal{V} \times \mathcal{V} \to \mathcal{T}$	traveltime from i to j
$mot_i : \mathcal{N} \to \mathcal{P}$	transport mode for nurse i

Table 1. Parameters of the CP model for the MHS problem

4.1 Problem Parameters

The problem parameters are given as specified in Section 2 and are summarized in Table 1. We denote M_u the number of *unfixed* jobs (where nurse and time slot are not yet set) and M_f the number of *fixed* jobs (where nurse and timeslot are fixed, e.g. appraisal interviews), such that the number of jobs $M = M_u + M_k$. Note that the set of jobs is ordered and contains all unfixed jobs first, followed by fixed jobs.

For our constraint model, we additionally introduce the set of visits $\mathcal{V}$, i.e. all stops that are part of a tour. The visits contain all jobs, as well as the trips from the nurses' home to the first job, denoted *start-visit*, and the trips from the last job to the nurses' home, denoted *end-visit*, hence yielding a total of $M+2N$ visits. We summarise the start-visits in the set $\mathcal{S} = \{M \dots M+N-1\}$ and the end-visits in $\mathcal{E} = \{M+N \dots M+2N-1\}$. Those visits mark the start and end point of each tour. Hence, the set of visits is composed of $\mathcal{V} = \mathcal{J} \cup \mathcal{S} \cup \mathcal{E}$ and ordered in that sense (first the jobs, then start-visits, followed by end-visits): $\mathcal{V} = \{0, \dots, M, \dots, M+N, \dots, M+2N-1\}$.

We represent time by a set of discrete points in time, defined by $\mathcal{T} = \{0..T\}$ where 0 represents 00:00 and T the last point of time of the day. The time points are given in τ-minute intervals, where $\tau = 5$. In other words, $\mathcal{T} = \{0, 1, 2, \ldots, T\}$ represents the following points in time: {00:00, 00:05, 00:10,. . . ,23:55}.

The travel time from visit i to visit j using transport mode $p \in \mathcal{P}$ is computed and collected in the distance matrix $tt_{ij}^{P} \in \mathcal{T}$. This means, for instance, that an entry $tt_{ij}^{car} = 5$ states that the travel time from i to j by car takes 5 points in time, thus $5{*}\tau{=}25$ minutes. Note that we round the actual travel times up to the next point in time, e.g. 23 minutes travel time are rounded up to 25 minutes.

4.2 Variables

The variables in a constraint model capture the decisions that have to be made. We formulate the problem as a discrete model where the number of decisions has to be finite and is represented by the *domain* of the decision variable that contains all possible choices for the respective variable. We use the notation

$$var_k \in Dom \text{ where } k \in \{1..K\}$$

to define K variables labelled *var* that can take values specified in the domain *Dom*. Typically, *Dom* is a finite set of integer values. Note that the values representing the transport mode, aspects and qualification are integral values.

The first important set of variables represents the decision of which **nurse** performs visit j and hence

$$nurse_j \in \mathcal{N} \text{ where } j \in \mathcal{V} \tag{1}$$

defines a variable for each visit capturing which nurse performs visit j. In other words, the assignment $nurse_j = i$ states that nurse i performs visit j. Secondly, we represent the tours by introducing **predecessor** and **successor** variables that state which visit preceeds and succeeds another visit j:

$$succ_j \in \mathcal{V} \text{ where } j \in \mathcal{V} \tag{2}$$

$$pred_j \in \mathcal{V} \text{ where } j \in \mathcal{V} \tag{3}$$

Hence, the assignment $succ_j = i$ states that the successor of visit j is visit i. Third, we introduce variables t_j for each visit j to represent the **time** at which j is performed:

$$t_j \in \mathcal{T} \text{ where } j \in \mathcal{V} \tag{4}$$

Therefore, the assignment $t_j = k$ states that visit j takes place at time $k \in \mathcal{T}$. Furthermore, we use redundant 0/1 **helper variables** x_{ji} that state if job j is performed by nurse i.

$$x_{ji} \in \{0, 1\} \text{ where } j \in \mathcal{J} \text{ and } i \in \mathcal{N} \tag{5}$$

These variables are used for so-called 'channeling constraints' [8] that improve the model quality. Finally, we introduce variables that represent the **mode of transport** chosen for visit j:

$$tm_j \in \mathcal{P} \text{ where } j \in \mathcal{V} \tag{6}$$

4.3 Constraints

In this section we scetch the most important constraints in the CP model.

Initializing Constraints are the constraints that set up the initial state of the problem. For instance, the start and end visits of nurse i are performed by i:

$$\begin{aligned} \forall i \in \mathcal{N}.\ nurse_{i+M} &= i \\ nurse_{i+M+N} &= i \end{aligned}$$

Similarly, we initialize the predecessor/successor variables by setting the start and end of each tour: the nurse's home. In other words, the predecessor of 'leaving home' is 'coming home' and the successor of 'coming home' is 'leaving home':

$$\begin{aligned} \forall j \in \mathcal{S}.\ pred_j &= j + N \\ \forall j \in \mathcal{E}.\ succ_j &= j - N \end{aligned}$$

Furthermore, we initialize the x-variables by a simple channeling constraint:

$$\forall j \in \mathcal{J}.\ \forall i \in \mathcal{N}.\ x_{ji} = 1 \Leftrightarrow nurse_j = i$$

Similarly, we channel between successor and predecessor variables: the predecessor of the successor of visit j is j and vice versa. Channeling constraints reduce the search space and hence improve the problem model.

$$\begin{aligned} \forall j \in V. \quad & pred_{succ_j} = j \\ \forall j \in V. \quad & succ_{pred_j} = j \end{aligned}$$

Note, that in the expression $pred_{succ_j}$ the variable *pred* is indexed by another variable, namely $succ_j$. Such expressions are represented by a specific (non-linear) constraint, the so-called 'element constraint' *element*(x,y) $= z$ corresponding to $x_y = z$ where x, y and z are variables. Such expressions often appear in our constraint model and often need to be further decomposed into subconstraints (element constraints) and introduce auxiliary variables. However, for readabilty, we will use the notation x_y though the expression is actually represented by a more complex set of subexpressions.

Finally, we impose the constraints for the fixed jobs (those jobs that are already pre-allocated to a specific nurse and time slot):

$$\begin{aligned} \forall j \in \mathcal{J}_f.\ nurse_j &= fixn_j \\ \forall j \in \mathcal{J}_f.\ t_j &= fixt_j \end{aligned}$$

Job Time Window Constraints restrict the starting time of job j to the respective starting time window tw_j^J where $tw_j^J(1)$ represents the start time of the time window, and $tw_j^J(2)$ represents the end.

$$\begin{aligned} \forall j \in \mathcal{J}_u. \quad & t_j \geq tm_j(1) \\ \forall j \in \mathcal{J}_u. \quad & t_j \leq tm_j(2) \end{aligned}$$

Job Tour-Constraints assure that each job is performed exactly once and hence does not appear in different tours. This means there may *not* exist two distinct visits i and j that preceed or succeed the same visit k, i.e. all predecessors/successors variables must take different values. For this we use the *alldifferent* constraint where *alldifferent*(x) on k variables x_i with $i \in \{1..k\}$ states that the values assigned to the variables x must be different.

$$\begin{array}{l} \mathit{alldifferent}(succ) \\ \mathit{alldifferent}(pred) \end{array}$$

Nurse Tour Constraints state that all jobs in a nurse's tour must be performed by the same nurse. These constraints are separately imposed over the predecessor and successor variables: the nurse performing visit j must be the same performing the preceeding/succeeding visit of j.

$$\begin{array}{l} \forall j \in \mathcal{V}.\ nurse_j = nurse_{pred_j} \\ \forall j \in \mathcal{V}.\ nurse_j = nurse_{succ_j} \end{array}$$

Nurse Time Constraints assure that the jobs assigned to nurse i respect the nurse's working time windows tw_i^N. First, we state that the time at which visit j starts has to be greater or equal to the starting time of the nurse assigned to j's shift, plus the travel time from the nurse to j. Second, we state that the end of the nurse's shift has to be greater or equal to (before or at) the time visit j is scheduled, plus j's duration and the traveltime from j to nurse performing j.

$$\begin{array}{ll} \forall j \in \mathcal{V}.\ t_j \geq & tw^N_{nurse_j}(1) + tt_{nurse_j,j} \\ \forall j \in \mathcal{V}.\ tw^N_{nurse_j}(2) \geq t_j + dur_j + tt_{j,nurse_j} & \end{array}$$

Aspect Constraints For each job j, if j has a non-empty list of aspects, then for all nurses i that share an aspect a with job j, nurse i may not perform j.

$$\begin{array}{l} \forall j \in \mathcal{J}_u.\ (\mathcal{A}_j^C \neq \emptyset) \Rightarrow \\ \qquad \forall i \in \mathcal{N}.\ (\exists a \in \mathcal{A}_i^N.\ \ a \in A_j^C) \Rightarrow \quad nurse_j \neq i \end{array}$$

Transport Mode Constraints The transport mode for job j is the same as the transport mode of the nurse performing j.

$$\forall j \in \mathcal{V}.\ tm_j = mot_{nurse_j}$$

Typically, mot_{nurse_j} is represented by an element constraint (as mentioned above), however, since *mot* is a parameter and not a variable, we can also represent the constraint as an *extensional constraint* (also called 'table constraint') that is simply a list of feasible value tuples for a given set of variables. This representation is more efficient in our model when using the table constraint representation from Lecoutre [14] as implemented in JaCoP [13].

4.4 Search Strategy

The solving procedure in Constraint Programming has two core components: *propagation* and *search* [19]. While propagation filters the variables' domains according to the constraints (and their consistency level), search systematically assigns values to variables in an attempt to find a solution. Two main features characterize the search strategy: *variable* ordering and *value* ordering. The variable ordering states in which order decision variables are assigned a value during search (e.g. which variables to search upon first). On the other hand, the value ordering determines in which order values are assigned to a variable that has been chosen for search (e.g. which value should be assigned to a variable first).

Below we outline the search settings we have applied for the MHS. Using these settings, search is extremely effective: typically no more than 10% of the decisions made during search are wrong.

Variable Order. We apply a static variable order, where first, we fix half of the *successor* and *predecessor* variables, choosing every odd predecessor/successor. This means, we start with fixing parts of the job sequence (parts of the tours). Second, we set half of the *nurse* variables (every odd nurse), i.e. we assign nurses to jobs. Third, we set the remaining half of predecessor/successor variables, followed by the remaining half of nurses. Finally, we search for job start times t and transport mode for each job (the channeling variables x are redundant and hence not included into search).

Value Order. The value order for all variables is a simple smallest-value-first selection heuristic (i.e. the smallest values in the variable's domains are tried out first). For this purpose, we order the nurses by their start time (i.e. nurses available in the morning come first), and jobs by their desired start time. This way, we try to assign 'early' nurses and 'early' jobs first, followed by later jobs/nurses. This heuristic also assures that the job start times are as early as possible which omits unnecessary waiting times between jobs.

5 Problem Clustering

Decomposing the problem qualification-wise does not yield equally-sized subproblems. For instance, jobs of qualification 'basic homecare' typically make up 80% of all jobs, yielding again a far too large subproblem. We therefore introduce another decomposition step that is triggered for particular qualifications or if a time-limit is reached: decomposition into spatial clusters.

A cluster consists of a set of jobs and nurses from the overall problem instance. Finding a *feasible* cluster, where the underlying instance is solvable (in reasonable time) is not straightforward, since both *vicinity* and *temporal availability* are crucial. In other words, for all jobs within a cluster, we need to find a set of nurses who can reach the jobs within time (vicinity) and who are available in the jobs' time slots (temporal availability). In our approach, we create and solve clusters one by one, and *iteratively* alter those clusters that we cannot solve straight away.

5.1 Creating Clusters using a Quadtree

We reflect the spatial distribution of jobs by inserting them into a quadtree, where each node represents a job. A quadtree is a tree datastructure, whose nodes have up to four children that, for our purpose, are labeled *north-east*, *south-east*, *south-west* and *north-west* and are the roots of subtrees containing all nodes (jobs) whose GPS locations are *north-east*, *south-east*, *south-west* and *north-west* in relation to their parent node. Jobs are then recursively added (with random order) starting with an "one-node" tree whose root is located at a center location of Vienna, to assure that the quadtree is reasonably balanced.

Now we can initiate the selection. First, we select k jobs by k times removing the first leaf that we find using a simple depth-first search. In this way, the k selected jobs are reasonably closely located. Second, we select m nurses with $m = \min(k, N_{I_q})$ where N_{I_q} represents the number of available nurses for the subinstance. As we want to find nurses that are reasonably close to the jobs, we pick one of the selected jobs at random and choose the m nurses closest to it. This way, we retrieve a cluster with k jobs and m nurses.

5.2 Iteratively Altering Clusters

We now invest a certain amount of time τ_k trying to solve the obtained cluster. If we cannot find a solution within this time, we iteratively simplify the instance until we find a solution or reach an overall time-limit τ:

1. Given jobs $\mathcal{J}$, nurses $\mathcal{N}$, cluster-timeout τ_k and overall timeout τ
2. If $|\mathcal{J}| \geq 1$ and timeout τ has not yet been reached:
   create cluster $C = (\mathcal{J}', \mathcal{N}')$, where $\mathcal{N}' \subseteq \mathcal{N}$, $|\mathcal{N}'| = m$ and $\mathcal{J}' \subseteq \mathcal{J}$, $|\mathcal{J}'| = k$, otherwise stop.
3. If timeout τ has not yet been reached: attempt to solve cluster C in τ_k secs
   (a) If solving was successful, update the set of overall jobs $\mathcal{J}$ to $\mathcal{J} - \mathcal{J}'$ and the set of overall nurses $\mathcal{N}$ to $\mathcal{N} - \mathcal{N}'$ and go to 2.
   (b) Otherwise, if $|\mathcal{J}'| \geq 1$, remove $f(\mathcal{J})$ jobs from $\mathcal{J}'$, yielding a new, smaller cluster $C = (\mathcal{J}'', \mathcal{N}')$ and go to 3.
   (c) Otherwise, if $|\mathcal{N}| \geq 1$ add l nurses to $\mathcal{N}'$, yielding a new cluster $C = (\mathcal{J}', \mathcal{N}'')$ and go to 3.
   (d) Otherwise, increase τ_k by δ time units

In our current setup, we use k=25, τ_k=1s, τ=300s, δ=10s, $f(\mathcal{J})$=$|\mathcal{J}|$-10 if $|\mathcal{J}| \geq 10$ and $|\mathcal{J}|$-1 otherwise, l=1. This setup typically produces a valid solution for a 700 jobs/500 nurses instance within 15–20 seconds; more details and experimental results can be found in Section 6. In case we cannot find an overall solution within τ seconds, the CP approach has failed and we use an alternative approach, as outlined in Section 3.1.

6 Experimental Results

In our empirical evaluation we assess two main aspects: (1) the initial solution generation and (2) the effects of valid initial solutions on the performance of the metaheuristics.

6.1 Problem Instances and Experimental Setup

We use a random selection of real-world instances from our project partner SoGL, one of the biggest homecare services in Vienna. Please note, that we cannot publish the instances due to data protection regulations. The instances have the following dimensions (the number of nurses corresponds to those that are *available*—only some of them (30-40%) are employed in final solutions):

	day 1	day 2	day 3	day 4	day 5	day 6	day 7	day 8
# nurses	509	491	504	482	496	518	500	505
# jobs	711	700	679	682	708	699	717	679

Our framework has been implemented within the same framework in Java version 1.6 and JaCoP [13] has been used as constraint solver.

6.2 Initial Solution Generation

We start by evaluating the performance of the initial solution generation. All tests were run on a single core of a Intel(R) Core(TM)2 Quad CPU Q9300 2.50GHz with (at most) 4GB RAM assigned. The heuristic setup is the same as described in Section 5.

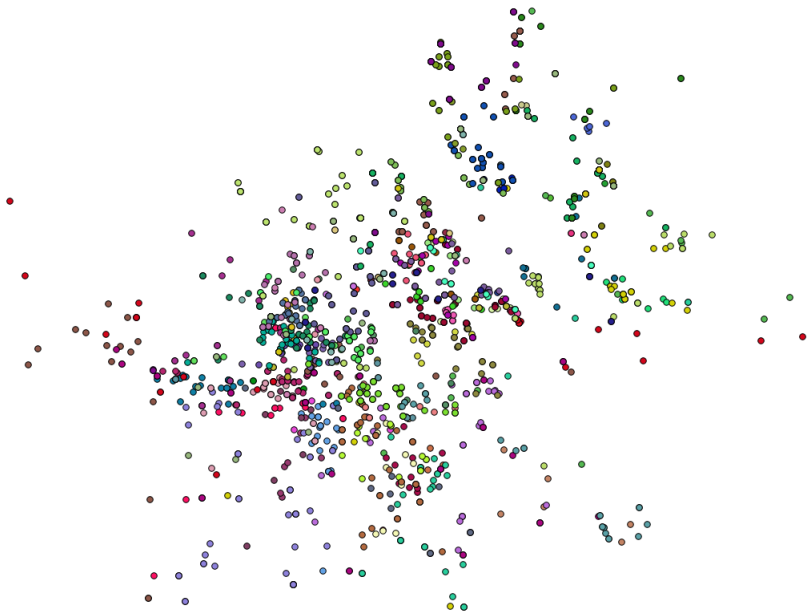

Fig. 1. Spatial (solved) clusters for the 'homecare' qualification for cluster-size 15

Figure 1 shows the spatial distribution of 'basic homecare'-jobs for cluster size 15, where each cluster is represented by a colour. We can see that the jobs within a cluster are fairly close together, but clusters overlap a lot (recall that the cluster size corresponds to the number of jobs in a cluster).

The cluster size has a strong impact on both the solving process and the generated solutions. We summarize the main aspects in Figure 2 where we compare instances with a distance matrix of *estimated* travel times to instances with *fixed* travel times, where the distance between every location is set to 15 minutes (which is the measure that is currently used in the health care company).

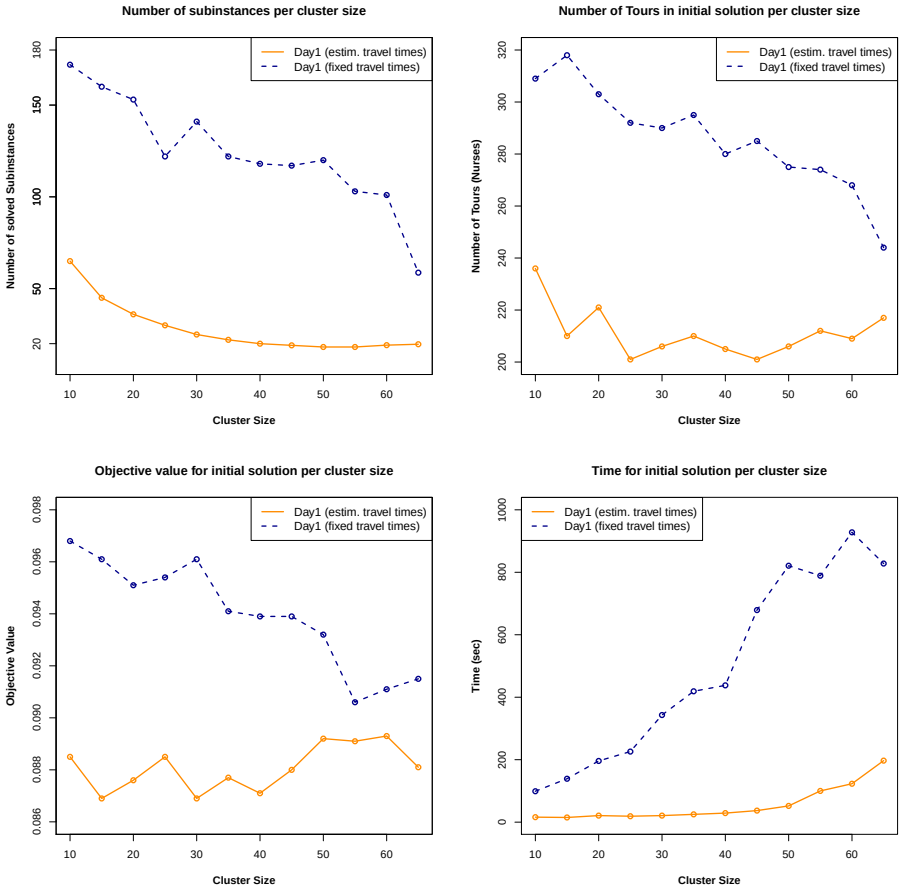

Fig. 2. Initial solutions depending on cluster size: the number of subinstances, i.e. generated clusters (top left) the number of tours in a solution (top right), initial objective value (bottom left) and solving times (bottom right)

Clustering Performance First, we consider the degree of decomposition, reflected by the *number of subinstances* into which the problem is decomposed to. A high number of subinstances results from many iterations during the clustering approach: if a cluster of a given size n is not solvable within a time-limit, n is iteratively reduced until a solution can be found. Hence, many iterations result in many small clusters.

The number of subinstances per clustersize is given in Figure 2(top, left), where the dotted blue line represents the results for *fixed* travel times and the yellow line the results for *estimated* travel times. First, we see that the number of subinstances (slightly) decreases with increasing cluster size, as expected. Moreover, with fixed travel times, the clustering approach yields far more subinstances (>100 on average) than for estimated travel times (25 on average). Hence, the CP model obviously struggles to find a valid solution within the time-limit. One reason for this is probably that estimated travel times can 'guide' CP search towards a solution, (short travel times are preferred through the search settings), while fixed travel times do not eliminate options, since all options (trips) have the same length.

We also observe another phenomenon as a result of a high number of subinstances: since the resulting clusters are particularly small (e.g. 10 jobs and 7 nurses), the resulting solutions of the clusters only assign very few jobs to each nurse, especially since the time windows of the jobs are not taken into account during spatial clustering. Therefore, the generated tours are particularly short and far too many nurses are employed. This is reflected in the *number of tours* that states how many nurses are employed in the solution (one nurse per tour) and which is depicted in Fig. 2 (top right). There we see that for fixed travel times, the number of tours is much higher (between 250-320) than that for estimated travel times (mainly 200-220).

Solution Quality The solution quality is analyzed in Fig. 2 (bottom, left): the solutions using estimated travel times have a substantially higher quality (≈0.88) than those using fixed travel times (≈ 0.94). One main reason is the high number of tours in the fixed travel times solutions: A high number of tours in the solution represents a roster with a low nurse workload and is hence penalized by the objective function. Using estimated travel times, we generate tours with much higher workload, where nurses are assigned to far more jobs per tour. Furthermore, we see that for the estimated travel times case, the cluster size only mildly impacts the solution quality .

Solving Time Figure 2 illustrates the solving time depending on the cluster size. We observe that construction time increases with cluster size, in case of fixed travel times, even quite dramatically. This mainly results from the computatinal overhead that results when a cluster of a given size cannot be solved straight away and additional iterations to reduce the cluster are required.

In summary, we see that we greatly benefit from using estimated travel times and that using a fairly small cluster size yields relatively good initial solutions in

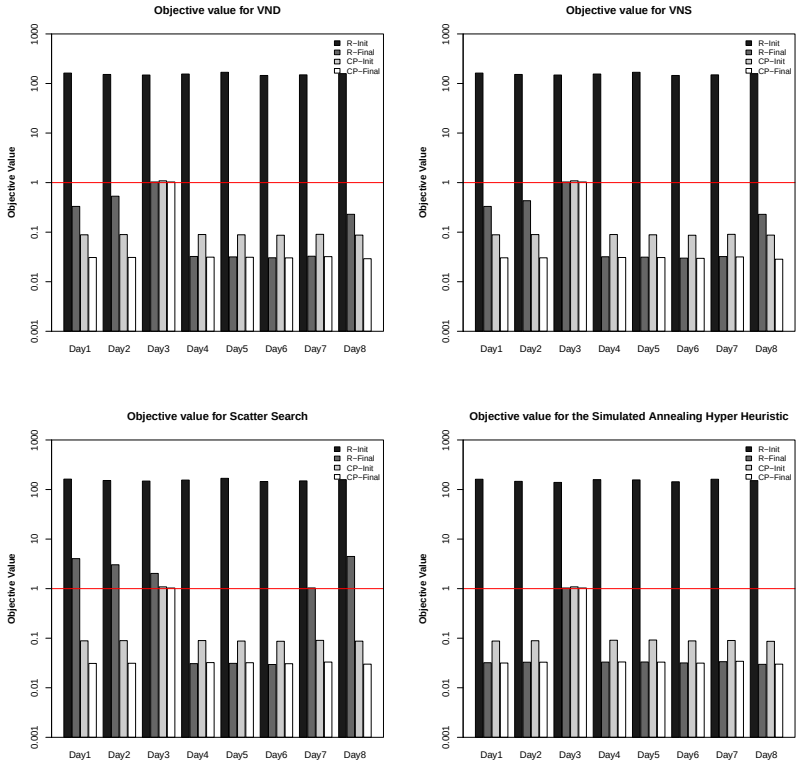

Fig. 3. Initial and final objective values using CP and random construction. Note that day3 is not solvable due to a data error.

short computation times (approx. 20 seconds). Therefore, for our experimental evaluation in combination with metaheuristics we choose a default cluster size of 20 with estimated travel times.

6.3 Optimizing with valid initial solutions

For evaluating the impact of valid initial solutions on the proposed metaheuristics, we performed experimental tests where each one is limited to a maximum of 75 minutes to improve the initial solution. The initial solution is either generated via CP or via the random construction heuristic. Note that in this paper we only consider metaheuristics that make proper use of a valid initial solution. Therefore other approaches, such as evolutionary algorithms that take a large population of initial solutions (of which most are typically invalid) and for which therefore a valid initial solution has little effect, are not taken under consideration.

The parameter settings of each metaheuristic are summarized in Table 6.3. We perform 10 runs, each on a single core of a 8xDualcore AMD Opteron 870

SAHH Parameters	**Settings**
K (# iterations)	10000
acceptance prob. (start)	0.05
acceptance prob. (end)	0.005
learn period	$K/500$
nrep (#iter/temp)	#neighborhoods
min weight	0.1
change of temperature t	Lundy and Meers' nonlinear function $t = t/(1+\beta t)$, $\beta = ((t_{\text{start}} - t_{\text{end}}) \cdot nrep/K \cdot t_{\text{start}} \cdot t_{\text{end}})$

SS Parameters	**Settings**
RefSet size	5
combination operator	path-relinking
s_{max}	100
time-limit LS	10sec
termination	time limit or no improvement after an iteration

Table 2. Parameter settings for the SAHH and SS

2GHz with a maximum of 4GB RAM (per run). All techniques are implemented within the same framework in Java. In Fig. 3. shows results for the final objective values, depending on the construction heuristics, averaged over 10 runs. Furthermore, Fig. 4 illustrates how the solution quality is improved in each metaheuristic over time: Fig. 4(left) using a randomly constructed, invalid initial solution and Fig. 4(right) using the initial solution from the CP-based heuristic. We observe a considerable benefit from using valid initial solutions, since all metaheuristics converge far quicker towards the final result and often yield considerably better results than when using a random initial solution.

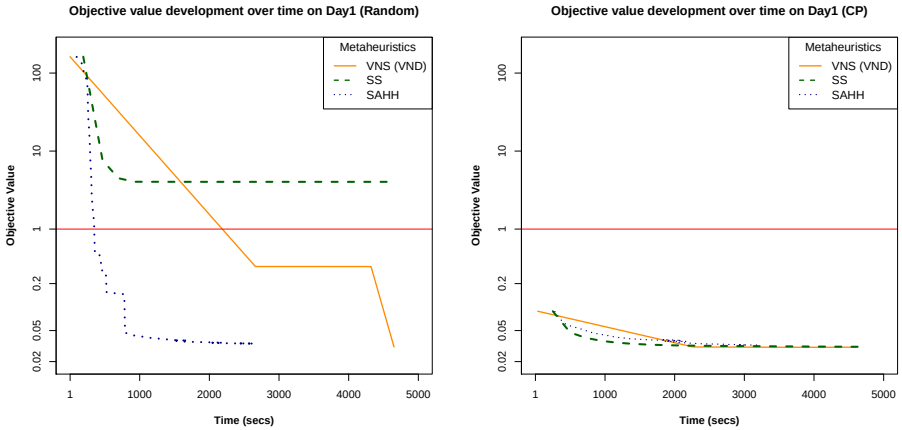

Fig. 4. Objective value development over time per metaheuristic, using either the random (left) or CP-based (right) construction heuristic. The objective values represent *means* over 10 runs (and hence sometimes overlap) and have not been sampled over time, but over particular stages in the approach (e.g. particular number of iterations).

The VNS approach produces the best results but does not converge as quickly as the SAHH or Scatter Search in the early search stages. It particulary benefits from the valid initial solution, where it converges far quicker towards the final result, saving almost 50% in time. The VND approach, though particularly simple, yields fairly good results in little time. Similar to the VND, it performs better with a valid initial solution. Note, that the results of the VND approach correspond to the results of the first iteration of the VNS approach in Fig. 4.

The Simulated Annealing Hyper Heuristic converges quickly towards a better solution but does not manage to further improve it. The valid initial solution has little effect on the performance of the SAHH, probably since the SAHH is designed to accept bad solutions during its early iteration stages.

The Scatter Search approach benefits greatly from the hybrid setup: if initiated on valid solutions from the CP-heuristic, it produces solutions of better quality than the SAHH. However, if initiated with an invalid solution, it often fails to even produce valid solutions after the improvement phase.

7 Conclusions

In this work, we tackle a large-scale real-world Multimodal Homecare Scheduling (MHS) problem using a CP-based construction heuristic combined with one of five alternative metaheuristic approaches. First, we show how to generate a valid initial solution using a Constraint-Programming-based clustering heuristic that produces results in very little time. Second, we demonstrate the positive impact of valid initial solutions on the performance of different metaheuristics in our empirical study. In summary, this work demonstrates the potential of hybrid approaches for tackling large-scale real-world MHS problems.

In future work, we plan to extend our current one-day approach to generate multi-day solutions, where we also consider shift and working hour constraints that hold over a longer period of time. Furthermore, we want to explore very large neighborhood search [1] as another local search procedure to further improve the results obtained so far for the MHS problem.

References

1. Ahuja, R., Orlin, J., Sharma, D.: Very large-scale neighborhood search. International Transactions in Operational Research 7(4-5), 301–317 (2000)
2. Bai, R., Blazewicz, J., Burke, E.K., Kendall, G., Mccollum, B.: A simulated annealing hyper-heuristic methodology for flexible decision support. Tech. rep., School of Computer Science, University of Nottingham, England (2006)
3. Bertels, S., Fahle, T.: A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. Comput. Oper. Res. 33, 2866–2890 (October 2006)
4. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science 39(1), 104–118 (2005)

5. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part ii: Metaheuristics. Transportation Science 39(1), 119–139 (2005)
6. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A scatter search approach to the nurse rostering problem. Journal of the Operational Research Society 61(11), 1667 – 1679 (2010)
7. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. Journal of Scheduling 7, 441–499 (November 2004)
8. Cheng, B.M.W., Choi, K.M.F., Lee, J.H.M., Wu, J.C.K.: Increasing constraint propagation by redundant modeling: an experience report. CONSTRAINTS 4, 167–192 (1999)
9. Eveborn, P., Flisberg, P., Ronnqvisb, M.: Laps care - an operational system for staff planning. European Journal of Operational Research 171, 962–976 (2006)
10. Gent, I., Walsh, T.: Csplib: a benchmark library for constraints. Tech. rep., Technical report APES-09-1999 (1999), available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).
11. Glover, F., Laguna, M., Mart, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics 39, 653–684 (2000)
12. Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover, F.W., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer Academic Publisher, New York (2003)
13. Krzysztof, K., Szymanek, R.: Jacop java constraint solver (Dec 2011), `http://www.jacop.eu`
14. Lecoutre, C.: Optimization of simple tabular reduction for table constraints. In: CP. pp. 128–143 (2008)
15. Nikolaev, A.G., Jacobson, S.H.: Simulated annealing. In: Gendreau, M., Potvin, J.Y., Hillier, F.S. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, pp. 1–39. Springer (2010)
16. Prandtstetter, M., Raidl, G.R., Misar, T.: A hybrid algorithm for computing tours in a spare parts warehouse. In: Cotta, C., Cowling, P. (eds.) Evolutionary Computation in Combinatorial Optimization - EvoCOP 2009. LNCS, vol. 5482, pp. 25–36. Springer (2009)
17. Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J.: The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. Tech. Rep. 11-2010, DTU Management Engineering (May 2010)
18. Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J.: The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research 219(3), 598 – 610 (2012)
19. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
20. Steeg, J., Schröder, M.: A hybrid approach to solve the periodic home health care problem. In: Kalcsics, J., Nickel, S. (eds.) Operations Research Proceedings 2007, Operations Research Proceedings, vol. 2007, pp. 297–302. Springer Berlin Heidelberg (2008)