Hybrid Heuristics for Multimodal Homecare Scheduling

Andrea Rendl, Matthias Prandtstetter, Gerhard Hiermann, Jakob Puchinger, Günther Raidl

Hybrid Heuristics 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. We focus on hybrid solution methods for a large-scale real-world multimodal homecare scheduling (MHS) problem, where the objective is to find an optimal roster for nurses who travel in tours from patient to patient, using different modes of transport. In a first step, we generate a valid initial solution using Constraint Programming (CP). In a second step, we improve the solution using one of the following metaheuristic approaches: (1) variable neighborhood descent, (2) variable neighborhood search, (3) an evolutionary algorithm, (4) scatter search and (5) a simulated annealing hyper heuristic. Our evaluation, based on computational experiments, demonstrates how hybrid approaches are particularly strong in finding promising solutions for large real-world MHS problem instances.

1 Introduction

The demand for care of the elderly is constantly increasing, in particular in today's western world, thus efficient health care services are of great significance. The idea behind home health care is to nurse patients 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, using specific transport modes. The goal is to find a nurse roster where all jobs are assigned to adequate nurses and employer, nurse and customer satisfaction is maximized.

Finding a good nurse roster is challenging, since the problem is a combination of two NP-hard problems: vehicle routing with time windows (VRPTW) [5, 6] and nurse rostering (NRP) [8]. Moreover, we consider a *large-scale real-world* setup from a Viennese public health care company. Therefore, we apply a *heuristic* approach since exact methods are typically too expensive for large instances. The idea is to start with an initial valid solution that is then systematically improved by different metaheuristic approaches that we evaluate in this work. Another aim is to construct a *flexible framework* that can be applied in other homecare scenarios with similar side constraints. Therefore, our approach is as generic as possible, where side constraints can be relatively easily altered. We

start by defining the problem (Section 2), then present the initial solution generation (Section 3) and the metaheuristics (Section 4), discuss computational results (Section 5) and conclude our work in Section 6.

2 The Multimodal Homecare Scheduling Problem

The Multimodal Homecare Scheduling (MHS) problem deals with finding a roster for nurses who perform nursing services at patients' homes. Therefore, the problem contains a *rostering* component (assigning nurses to jobs, w.r.t. various shift/working hour restrictions), as well as a *vehicle routing* component (nurses travel in tours from patient to patient, starting at their home). The multimodal aspect arises from the transport mode each nurse chooses for traveling.

Basically, we have a set of customers, the *patients*, who book one or more jobs, daywise summarized in the set of jobs $\mathcal{J} = \{1, \ldots, J\}$. Each job has an associated *location*, given as GPS coordinates, a *start time window* in which the job should start, and a *duration*. Furthermore, each job requires a minimal *qualification (level)* $q \in \mathcal{Q}$ from whoever performs the job (e.g. taking blood samples requires a medical qualification). In our problem setup, $|\mathcal{Q}| = 5$, i.e. there are 5 different qualifications.

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* of each nurse as GPS coordinates. Each nurse has a *qualification (level)* $q \in \mathcal{Q}$ and may only perform jobs that require a qualification that is less or equal the nurse's maximal qualification. Hence, we have an ordering relation defined on the qualifications, and nurses of higher qualification may perform lower-qualified jobs, but not vice versa. Nurses can only work within specified *time windows* that vary from day to day, and have a minimal and maximal *number of working hours* per day.

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

A novel aspect in this work is the consideration of *multimodality*: each nurse states her preferred *mode of transport* (public transport or car) 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 and jobs 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 it is then desired to select that 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 or preferences for nurses)
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:

- 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.
- Availabilities of nurses have to be regarded, i.e. jobs can only be assigned to nurses whenever they are available.
- Pre-allocated jobs must not be shifted to other times or nurses.

On the other hand, the following requirements are only soft, i.e. violations of them are not welcome but tolerated (with corresponding penalization in the objective function):

- 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.
- 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.
- Preferences stated by nurses and customers are handled as soft constraints, since all jobs have to be accomplished (even if preferences are violated).
- 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).

– 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

The home health care problem has been considered using different approaches: Eveborn et al. [9] choose a set-partitioning formulation for the problem, and primarily focus on the application of *repeated matching*. Bertels and Fahle [4] combine *linear programming* (LP) for computing optimal start times, *constraint programming* (CP) for generating initial solutions, and *simulated annealing* (SA) and *tabu search* (TS) based heuristic for subsequent solution improvement. Steeg and Schröder [20] use CP to construct an initial solution that is improved by a combination of *Adaptive Large Neighborhood Search* (ALNS) and SA. Rasmussen et al [18] use a set partitioning formulation of the problem and apply a specialized branch-and-price solution algorithm.

The main difference of our work to other approaches lies within (1) the novel consideration of multimodality, (2) our objective to provide a *flexible* framework to tackle real-world MHS problems with varying side constraints (3) a novel, broad selection of metaheuristics for the MHS problem and (4) the excessively large size of instances that we tackle.

3 Initial Solutions with Constraint Programming

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, expendable system.

The instances we consider are far too large to be solved within a single CP model, therefore we simplify the problem by solving each instance qualification-wise: 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. This means 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.

3.1 The CP Model and CP Search Setup

We formulate the MHS problem as an extension of the Vehicle Routing Problem with Time Windows (VRPTW) model from [19] that contains three kinds of variables: (1) predecessor and successor variables, $pred_i$ and $succ_i$ of visit i that capture the sequence of tours, (2) nurse variables n_i, stating which nurse performs visit i, and (3) variables t_i for the start time of each visit i. The model is extended with binary variables x_{ij}—a value of 1 indicates that nurse i performs job j—for channeling and add constraints to represent multimodality, nurse and patient preferences, shift length as well as redundant constraints to increase propagation.

We apply a static search heuristic, where we first set half of the successor and predecessor variables (i.e. we start with fixing the job sequence), then set half of the nurse variables (nurse to job assignments), followed by the remaining half of predecessor/successor variables and remaining nurses. Finally, we search for job start times. Nurses are ordered by their start time (i.e., nurses available in the morning come first), and jobs by their desired start time. This way, search is extremely effective with an ascending value selection (smallest value first): typically no more than 10% of the decisions made during search are wrong.

3.2 Iterative 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.

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.

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 $\mathcal{M}$, nurses $\mathcal{N}$, cluster-timeout τ_k and overall timeout τ
2. If $|\mathcal{M}| \geq 1$ and timeout τ has not yet been reached:
   create cluster $C = (\mathcal{M}', \mathcal{N}')$, where $\mathcal{N}' \subseteq \mathcal{N}$, $|\mathcal{N}'| = m$ and $\mathcal{M}' \subseteq \mathcal{M}$, $|\mathcal{M}'| = k$, 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 $\mathcal{M}$ to $\mathcal{M} - \mathcal{M}'$ and the set of overall nurses $\mathcal{N}$ to $\mathcal{N} - \mathcal{N}'$ and go to 2.
   (b) Otherwise, if $|\mathcal{M}'| \geq 1$, remove $f(\mathcal{M})$ jobs from $\mathcal{M}'$, yielding a new, smaller cluster $C = (\mathcal{M}'', \mathcal{N}')$ and go to 3.
   (c) Otherwise, if $|\mathcal{N}| \geq 1$ add l nurses to $\mathcal{N}'$, yielding a new cluster $C = (\mathcal{M}', \mathcal{N}'')$ 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, $f(\mathcal{M})$=$|\mathcal{M}|$-10 if $|\mathcal{M}| \geq 10$ and $|\mathcal{M}|$-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 5. 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 3.3.

3.3 Random Solution Generation

As an alternative approach, 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 (such as appraisal interviews) 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.

4 Solution Improvement by Metaheuristics

A solution to the MHS problem consists of a list of tours—one for each nurse—where a tour contains a sequence of jobs, or is empty in case the nurse is not scheduled for that particular day. We receive such a (valid) initial solution from the solution generation heuristics discussed in the previous section. However, such a solution is typically of rather poor quality as some aspects of the soft constraints in the objective function (such as short travel times) have not yet been considered appropriately, and therefore cannot be used for real-world rosters. We improve the solution using different metaheuristics that we discuss in this section.

4.1 Variable Neighborhood Descent

Based on the idea that global optima are locally optimal w.r.t. all (possible) neighborhoods, *variable neighborhood descent* (VND) [13] tries to systematically examine a predefined set of neighborhood structures N_i, with $1 \leq i \leq l_{max}$ whereupon l_{max} denotes the number of defined structures. To speed up the search, the neighborhood structures are ordered such that potentially smaller neighborhoods $N(x)$ for a given solution x are examined first. As soon as a local optimum is reached (within the i-th neighborhood of 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.

Within this work, we define three neighborhood structures implicitly via three different move types:

shift mission A shift mission move shifts one mission from one tour to another tour by searching the best matching position in that other tour.

reposition mission When applying a move of that type, one mission is repositioned within its tour, i.e. the best matching position (without reordering the other missions) is determined.

swap nurses By this move, two nurses are swapped with each other, i.e. the tour of the first nurse is then handled by the second nurse and vice versa.

Due to the extensive size of the neighborhoods, first improvement is applied as step function such that improvements can be gathered rather fast. Although the initial neighborhood order (shift missions, swap nurses, reposition mission) leads to rapid improvements during the starting phase of VND, preliminary tests revealed that dynamic neighborhood reordering as applied in [17] leads to more promising results.

4.2 General Variable Neighborhood Search

One of the major drawbacks of VND is the fact that the search might get stuck in local optima. To overcome this, VND is typically embedded as local search phase in a general *variable neighborhood search* (VNS) scheme [13]. Here, the

Table 1. Operator setup for the hybrid Evolutionary Algorithm (MA)

EA Operator	Settings
selection	binary tournament
recombination	list crossover
mutation	move all unfixed missions to best other nurse
replacement	replace the worst similar solution in the population pool
improvement	cyclic search of neighborhoods - reorder mission neighborhood (best of improvement) - shift mission neighborhood (best of improvement) - swap nurse neighborhood (best of improvement)

basic idea is to provide diversification in order to escape from local optima by applying random moves in a set of larger neighborhoods. Performing these diversification moves also is 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 mission moves within the i-th consecutive iteration of VNS without improvement.

4.3 A Hybrid Evolutionary Algorithm

Evolutionary algorithms (EA) [12] imitate biological evolution by adopting basic concepts of evolutionary mechanisms such as reproduction, selection, mutation and recombination to iteratively derive better solutions. During initialization, a pool of diverse parent solutions is generated, the so-called *population*. From this population, a new generation of solutions is bred by selecting parental solutions according to a *selection procedure* that typically prefers solutions of higher quality and deriving offsprings with *recombination* and/or *mutation* operators, depending on specified probabilities. Recombination combines features of parents, while mutation typically performs small random alterations. Subsequently, the resulting offsprings replace parents in the population according to a *replacement strategy*. In this manner, the population is continuously evolving until a termination criteria is reached (e.g. timeout or a generation limit).

In this work we apply a hybrid EA, or Memetic Algorithm (MA) [15], that uses local search to improve the offspring before replacement. Since Burke et al [3] presented promising results for the Nurse Rostering Problem (NRP) with an MA, we expected similarly good results for our problem. In the following we discuss details about recombination, mutation and local search of our approach. The overall parameter settings are summarized in Table 1.

Initialization. The Random Diverse Constructor creates a solution similar to the random constructor described in section 3.3: instead of assigning a job to a nurse at total random, the constructor maintains a memory of already used assignments to create a new solution most different from the already constructed ones. For each job j, we select the nurse that has been least often assigned to j in already generated solutions and add j to the nurse's tour. If there is

more than one such nurse, we pick one randomly. The constructor memory is initialized with the starting solution (CP or random) by increasing the count of the corresponding job to nurse assignments. This way, the initial population is diverse, but is very likely to contain many invalid solutions.

Recombination Operator. Since solutions are lists of tours (one for each nurse), we implement a special recombination operator: given two parent solutions (P_1, P_2), the offspring is initially set to have the same tours as P_2. Then one nurse of P_1 is selected at random (n_i). Each unfixed job of n_i assigned in P_1 is removed in the offspring solution. Then every remaining job of n_i in the offspring solution is moved to the 'best' nurse n_j, i.e. where $n_j \neq n_i$ and which yields the lowest objective value. In the final step, the jobs of n_i in P_1 are assigned to n_i in the offspring solution.

Mutation. To provide higher diversity during the search of the MA, the offspring is mutated by selecting a nurse n_i at random and reassigning all unfixed tours of this nurse to other nurses n_j, where $n_j \neq n_i$, maximizing the objective value.

Local Search. For a given probability, a local search heuristic tries to further improve the offspring. To achieve a better balance between exploration and exploitation, the heuristic aborts after a certain time limit. We apply two local search algorithms: variable neighborhood descent (VND), as described in Section 4.1, and cyclic search of neighborhoods (CNS), which is similar to VND but always turns to the next neighborhood structure in a cyclic manner when a local optimum in one neighborhood structure or a time limit has been reached. In most of our experiments, CNS yielded better results than standard VND. The population is replaced using a slightly changed steady-state approach [21], where an offspring always replaces the most similar solution in the current population that is worse than the offspring. To calculate the similarity of two solutions, we count matching job to nurse assignments.

4.4 Simulated Annealing Hyper-Heuristic (SAHH)

Simulated Annealing (SA) [16] is an extension of local search that is inspired by the physical "annealing" process, a heat treatment of material in metallurgy, where a controlled heating and cooling process allows atoms to find new positions yielding a configuration of (closely) minimal internal energy. In contrast to basic local search, Simulated Annealing not only accepts better solutions for continuing the search but sometimes also worse. This is done in a probabilistic way considering the solution's objective value and a temperature parameter T, which is reduced over time. At the beginning, when T is large, accepting worse solutions is more likely, while later on when T has been decreased, almost only improvements are accepted. A Simulated Annealing Hyper Heuristic (SAHH) utilizes other local search techniques (low-level heuristics) as slaves to improve newly-constructed solutions. 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.

Bai et al [2] describe this kind of algorithm for the Nurse Rostering Problem (NRP). As the NRP is related to our problem, we considered this approach too. Starting from Bai et al.'s SAHH, the only changes that had to be made are the adequate replacement of the low-level heuristics and an appropriate selection of the total number of iterations. As low-level heuristics we apply variants of a simple local search procedure, whereupon the variants differ by the step functions applied and move types utilized. Beside a classical next improvement strategy we also incorporate a random-improvement strategy where we only guarantee that the number of (hard) constraint violations is not increasing (i.e. the integral part of the objective value must not increase). As move types we use the three previously defined move types, cf. Section 4.1. Thus, leading to a total number of six low-level heuristics.

Bai et al. suggest that around 10% of the solutions should be accepted at the beginning and only 0.5% at the end of the search, and we calculated our starting/ending temperatures also accordingly.

4.5 Scatter Search

Another population based approach that we evaluate is scatter search [11]. As described by Burke et.al. [7], this metaheuristic creates also good solutions for the NRP. 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.

Initialization. We create the initial population the same way as in the hybrid EA, where we use the solution found by the CP-Solver or random construction heuristic and derive additional solutions thereof.

Subset Generation. To generate subsets out of the pool of solutions (also called reference set or RefSet), a classical subset generation approach was used [11]: We first create subsets of all pairs of solutions. Next subsets containing of 2-sets plus the best solution not in the set are created (duplicate subsets are removed). At last, subsets of size $k = 4$ to the size of the RefSet containing the k best solutions are creates.

Subset Combination. For the combination of solutions for each subset we implement a path-relinking algorithm. Path-relinking [11] moves from a solution S_i to S_j where a move is defined by a different assignment of a variable. Unlike the classic path-relinking approach we do not perform a local search on a given number of promising solutions but create one solution out of the subset which will then be improved. To create a solution S_{new} out of a subset of size l we move iteratively from the worst solution S_1 to the best S_l. $S_{new} = S'_l$, where

$$S'_i = S'_{i-1} \rightarrow_{PR} S_i, i = 2, ..., l, S'_1 = S_1$$

and the path-relinking operator $S_i \rightarrow_{PR} S_j$ moves S_i $\mathit{difference}(S_i, S_j)/l$ steps to S_j. To determine which moves are performed first, we calculate the change

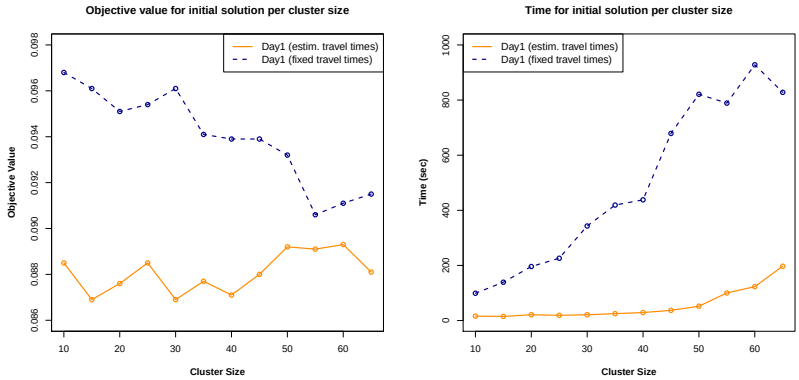

Fig. 1. Differences in initial objective values and solution times over varying cluster sizes.

in the objective per move and perform the move with the best change (which can also be worse). We restrict the number of moves to be calculated with a parameter s_{max} to prevent high runtimes.

This path-relinking combination algorithm was introduced after some preliminary tests with a 'construction by voting' approach described by Burke et al [7] that did not deliver promising results on our problem.

5 Computational Results

We assess our hybrid heuristics on a random selection of real-world instances from our project partner SoGL (that we cannot publish due to data protection regulations) with the same setup. All techniques are implemented within the same framework in Java version 1.6; JaCoP [14] has been used as constraint solver. The instances have the following dimensions (the number of nurses corresponds to those that are *available*—only about 35% are employed in 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

5.1 Initial Solution Generation Using CP

We analyze the impact of cluster size for initial solutions on *quality* (Figure 1, left) and *construction time* (Figure 1, right) using the heuristic setup as described in Section 3. We compare our *estimated* travel times to *fixed* travel times, where the distance between every location is set to 15 minutes (which is the travel time measure that is currently used in the health care company). All tests for this comparison where run on a single core of a Intel(R) Core(TM)2 Quad CPU Q9300 2.50GHz with (at most) 4GB RAM assigned.

Table 2. Final objective values for each metaheuristics, averaged over 10 runs. Note that instance 3 is unsolvable due to a data error.

	constr.	init.	VND	VNS	EA	SAHH	SS
day 1	cp	0.0885	0.03080 ±0.00067	0.03038 ±0.00034	0.02744 ±0.00057	0.03156 ±0.00064	0.03097 ±0.00034
	rand.	162.1513	0.33083 ±0.48339	0.33064 ±0.48340	0.02707 ±0.00031	0.03200 ±0.00080	4.03197 ±0.00065
day 2	cp	0.0894	0.03090 ±0.00039	0.03038 ±0.00028	0.02794 ±0.00037	0.03281 ±0.00122	0.03120 ±0.00050
	rand.	152.1521	0.53162 ±0.52725	0.43109 ±0.51647	0.02767 ±0.00037	0.03276 ±0.00094	3.03098 ±0.00055
day 3	cp	1.0860	1.02879 ±0.00055	1.02837 ±0.00044	1.02600 ±0.00051	1.03012 ±0.00095	1.02931 ±0.00051
	rand.	148.1494	1.02895 ±0.00043	1.02851 ±0.00040	1.02553 ±0.00036	1.03027 ±0.00078	2.02875 ±0.00048
day 4	cp	0.0896	0.03132 ±0.00044	0.03083 ±0.00048	0.02874 ±0.00042	0.03313 ±0.00062	0.03220 ±0.00051
	rand.	155.1532	0.03232 ±0.00083	0.03188 ±0.00087	0.02784 ±0.00056	0.03295 ±0.00082	0.03069 ±0.00040
day 5	cp	0.0885	0.03117 ±0.00052	0.03076 ±0.00050	0.02850 ±0.00044	0.03291 ±0.00091	0.03202 ±0.00077
	rand.	168.1506	0.03167 ±0.00057	0.03131 ±0.00068	0.02805 ±0.00047	0.03310 ±0.00081	0.03106 ±0.00048
day 6	cp	0.0867	0.03028 ±0.00055	0.02964 ±0.00060	0.02665 ±0.00048	0.03142 ±0.00120	0.03057 ±0.00068
	rand.	145.1523	0.03040 ±0.00051	0.02992 ±0.00056	0.02646 ±0.00025	0.03168 ±0.00153	0.02947 ±0.00043
day 7	cp	0.0904	0.03209 ±0.00049	0.03165 ±0.00054	0.02947 ±0.00035	0.03420 ±0.00104	0.03300 ±0.00050
	rand.	149.1516	0.03270 ±0.00085	0.03218 ±0.00080	0.02900 ±0.00042	0.03358 ±0.00093	1.03271 ±0.00049
day 8	cp	0.0872	0.02914 ±0.00071	0.02846 ±0.00056	0.02576 ±0.00038	0.02994 ±0.00061	0.02995 ±0.00046
	rand.	158.1446	0.22922 ±0.63228	0.22893 ±0.63227	0.02520 ±0.00041	0.02969 ±0.00064	4.46026 ±0.53455

First, we notice a large performance gap between the estimated and fixed travel time setup. One reason for this is that estimated travel times can "guide" CP search towards a solution since short travel times are preferred through the search settings. Furthermore, there is also a difference in solution quality, in particular for small cluster sizes. For estimated travel times, the solution quality is fairly similar, where smaller cluster sizes yield slightly better results. Finally, we also observe that construction time increases with cluster size, in case of fixed travel times, even quite drastically. 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 short computation times (approx. 20 seconds).

5.2 Evaluating Metaheuristics

For evaluating the proposed metaheuristics we performed experimental tests whereupon each one is limited to a maximum of 75 minutes to improve the initial solution, which is either generated via CP or via the random construction heuristic. The concrete parameter settings of each metaheuristic are summarized in Table 3. We perform 10 runs, each on a single core of a 8xDualcore AMD Opteron 870 2GHz with a maximum of 4GB RAM (per run). All techniques

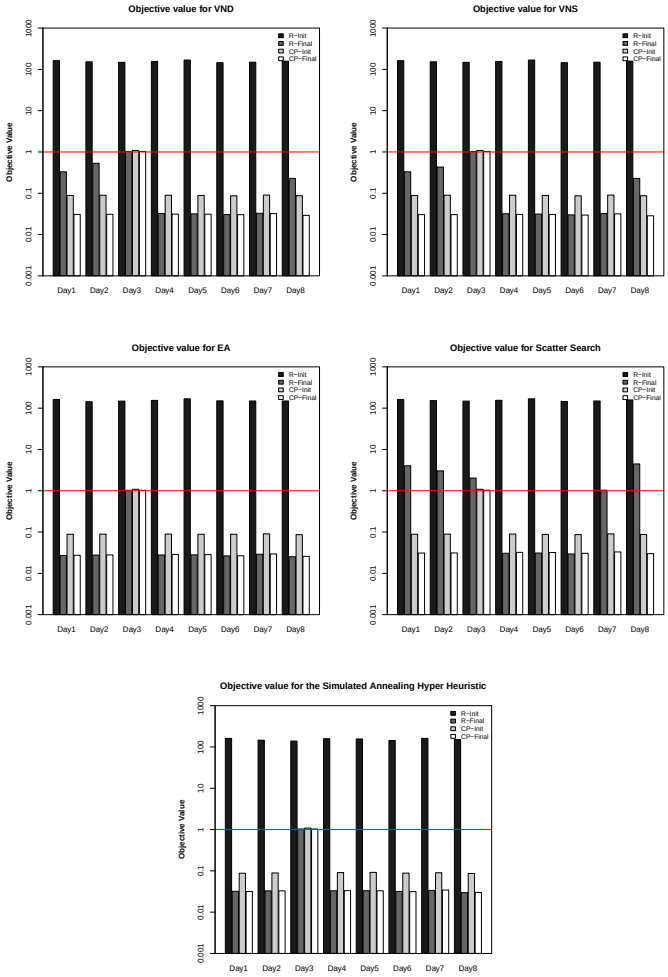

Fig. 2. Initial and final objective values using CP and random construction.

are implemented within the same framework in Java. In Table 2 we give results for the final objective values, depending on the construction heuristics, averaged over 10 runs, also illustrated in Fig. 2. Furthermore, Fig. 3 illustrates how the solution quality is improved in each metaheuristic over time: Fig. 3(left) using a randomly constructed, invalid initial solution and Fig. 3(right) using the initial solution from the CP-based heuristic.

First, we observe that the Evolutionary Algorithm produces the best results and is the quickest in converging. The initial CP-based solution has little effect, since the initial population consists of 99% randomly generated solutions. Therefore, the final results with the random construction heuristic are slightly better,

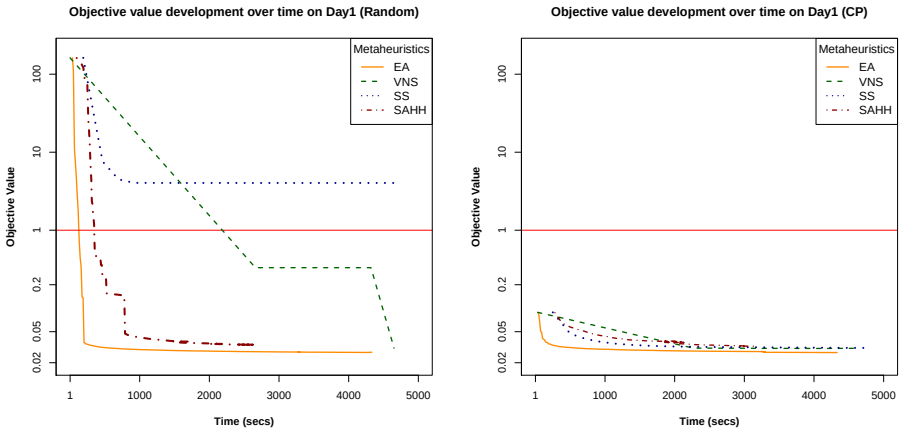

Fig. 3. Objective value development over time per metaheuristic, using either the random (left) or CP-based (right) construction heuristic. The objective values represent *means* over 10 runs (and hence sometimes overlap) and have not been sampled over time, but over particular stages in the approach (e.g. particular number of iterations).

since the EA had slightly more time to perform further iterations (instead of spending it on the CP construction heuristic).

The second best approach is the VNS approach that produces similarly good results as the EA, but takes more time to converge. We observe a considerable benefit from the combination with the CP construction heuristic: using the valid initial solution, the VNS converges far quicker towards the final result, saving almost 50% in time. Third comes the VND approach, that, 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. 3.

The Simulated Annealing Hyper Heuristic converges quickly towards a better solution but does not manage to further improve it. Similar to the EA, the 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.

In summary, we have seen that all metaheuristics produce satisfactory results in combination with the CP-based construction heuristic. Furthermore, VNS, VND and SS benefit from the hybrid setup, since the valid initial solution from the CP-based construction heuristic improves their performance. Moreover, since the CP-based construction is rather quick, none of the approaches suffers considerably from investing time in it.

Table 3. Parameter settings for EA, SAHH and SS

EA Parameter	**Settings**
population size	100
selection size	2
operator probabilities	
- recombination	1.0
- mutation	1.0
- improvement	0.01
termination	time limit or 2000 iterations without an improvement
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

6 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. Our contributions are threefold: first, we showed how we can generate valid initial solutions for the MHS in little time using a Constraint Programming-based clustering heuristic. Second, we studied the impact of different metaheuristics on a novel problem setup. Third, this work demonstrates the potential of hybrid approaches for effectively 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.

Acknowledgments This work is part of the project CareLog, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic programme I2VSplus under grant 826153. The authors thankfully acknowledge the CareLog project partners Verkehrsverbund Ost-Region GmbH (ITS Vienna Region), Sozial Global AG, and ilogs mobile software GmbH.

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. Bai, R., Burke, E.K., Kendall, G., Li, J., McCollum, B.: A hybrid evolutionary approach to the nurse rostering problem. IEEE Transactions on Evolutionary Computation 14(4), 580 – 590 (2010)
4. 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)
5. 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)
6. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part ii: Metaheuristics. Transportation Science 39(1), 119–139 (2005)
7. 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)
8. 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)
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. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 1st edn. (1989)
13. 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)
14. Krzysztof, K., Szymanek, R.: Jacop java constraint solver (Dec 2011), `http://www.jacop.eu`
15. Moscato, P.: Memetic algorithms: a short introduction, pp. 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999)
16. 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)
17. 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)
18. 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)

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)
21. Whitley, D., Kauth, J.: Genitor: A different genetic algorithm. Proc. of the Rocky Mountain Conf. on Artificial Intelligence pp. 118–130 (1988)