International Journal on Artificial Intelligence Tools

Assigning and Scheduling Service Visits in a Mixed Urban/Rural Setting

Mark Antunes[4],Vincent Armant[1], Kenneth N. Brown[1], Daniel Desmond[1], Guillaume Escamocher[1],Anne-Marie George[1], Diarmuid Grimes[1], Mike O'Keeffe[1], Yiqing Lin[3], Barry O'Sullivan[1], Cemalettin Ozturk[2], Luis Quesada[1], Mohamed Siala[1], Helmut Simonis[1] and Nic Wilson[1]

(1) Insight Centre for Data Analytics, School of Computer Science and Information Technology, University College Cork, Ireland
(2) UTRC-I, Cork, Ireland
(3) UTRC, East Hartford, CT, USA
(4) United Technologies Corporation, Winnipeg, Manitoba, Canada

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper[a] describes a maintenance scheduling application, which was developed together with an industrial partner. This is a highly combinatorial decision process, to plan and schedule the work of a group of travelling repair technicians, which perform preventive and corrective maintenance tasks at customer locations. Customers are located both in urban areas, where many customers are in close proximity, and in sparsely populated rural areas, where the travel time between customer sites is significant. To balance the workload for the agents, we must consider both the productive working time, as well as the travel between locations. As the monolithic problem formulation is unmanageable, we introduce a problem decomposition into multiple sequential steps, that is compatible with current management practice. We present and compare different models for the solution steps, and discuss results on datasets provided by the industrial partner.

Keywords: Maintenance Scheduling;Service Planning;Travelling repair person.

1. Introduction

Providing regular service checks and equipment repair of installed products is a key element of the overall business [5] for many industrial sectors. Service technicians have to visit equipment permanently installed at customer sites for testing, maintenance inspections, and repair work. The service requirements differ for different types of products, and may depend on a customer specific service level agreement.

[a]This works extends and improves an earlier model in [1]. The development of this paper was supported by United Technologies Corporation as part of a UCC Collaboration Project and by Science Foundation Ireland under Grant No. 12/RC/2289 P2, which is co-funded under the European Regional Development Fund.

On each day, a technician visits possibly multiple customer sites to perform planned and unplanned work. As the time spent travelling between customer sites is non-productive, a good schedule minimizes the amount of travel by selecting the best order of visits. While inside a metropolitan area the travel times can be insignificant, in rural areas travel may dominate the productive time spent on servicing equipment. Given due dates for planned work based on past visits should be enforced when possible, but this may conflict with the wish to minimize the amount of travel.

The team developing this application consisted of a mix of domain experts, managers of the business unit and end users, together with modelling experts, based on a close collaboration between academic and industrial partners. The system was developed from scratch for the specific client constraints and work procedures. Service quality is improved if technicians are familiar with individual device installations and with the customer, therefore the business requires that the same technician performs all work at a specific customer site, if that is possible. Following this rule, we came up with a natural decomposition of the problem into an assignment and a scheduling phase. The assignment phase decides which technician is responsible for which customer, while the scheduling component organizes the visits of the technicians to their assigned customer base. Not all technicians are trained to work on all types of equipment, and additional training may be required to cover unfamiliar devices. The service personnel may have preferences with regards to the amount of travel they have to do: some agents may prefer to only work within their home town during regular office hours, while others are willing to perform multi-day trips to more remote areas. Other preferences may concern the type of work required, or the locations for the first or last activity of the day, or indeed preferred time and place for lunch breaks.

We only consider deterministic task duration and travel times in the present work. This is easily justified for the balanced work assignment, which considers an aggregated yearly workload of planned and unplanned operations for each technician. To improve scalability of the approach and to reduce long-distance travel, we use a pre-clustering of customer locations, assigning customers in close proximity to the same cluster. We also try to minimize the diameter of each service area, and therefore the amount of intra-area travel, while at the same time balancing the workload of all technicians in a given depot location.

The mobile workforce scheduling problem combines aspects of vehicle routing[12] with personnel scheduling [8]. It is a well studied problem, see [5] for a survey. Some studies focus on a specific industry, for example electricity [10], water supply [4,28], copier repair [26] or elevator maintenance [20,3,29]. Others consider specific solution techniques, like large neighbourhood search [6,17] or genetic algorithms [22]. Heching and Hooker suggest the use of Benders Decomposition to solve a related problem of home-care scheduling [13], which was also considered in [23,14], but solved using meta-heuristics.

Mobile workforce scheduling can have a large impact on operational cost: A

tool for the mobile workforce management of BT Telecom is described in [18,19]. It combines constraint programming with optimization to handle a multi-skilled workforce of 20,000 technicians, with up to 150,000 tasks per day. A different IT platform for dispatching UPS drivers is described in [15]. The tool provides optimized routes for all 55,000 drivers collecting and delivering packages for UPS in North America. It is expected to save $300M to $400M per year in operational cost.

The main contributions of this current paper are the inclusion of planned and unplanned tasks, together with the travel between sites, in an overall capacity model for a planning period of one year, aggregating customers in close proximity by a pre-clustering to reduce problem complexity, and combining both single day and multi-day tours in a scheduling solution, solved with different models.

The main difficulty in comparing approaches presented in the literature is that in the various problem scenarios different elements of the problem dominate the others. In meter reading and parcel delivery the tour finding problem dominates all other aspects, while in some maintenance scheduling problems travel times can be neglected, and the medium/long-term problem of periodic scheduling dominates the problem. In our case study from industry, repair times and travel times are balanced, as customer sites are both closely spaced in urban areas, and far apart in rural parts of the service area.

The paper is structured as follows: We begin with a description of the overall problem decomposition (Section 2). We then describe the different stages of our solver, first a clustering method (Section 3), then a discussion of Route Generation (Section 4), a refinement of Aggregated Route Generation in Section 5, a local search based version of the Route Generation in Section 6, and finally, the scheduling model in Section 7. This is followed by an evaluation based on real-world end-user data in Section 8.

2. Overall Decomposition

In an ideal situation, one would find the daily schedule for all technicians, together with their yearly work assignment, in a single model, which optimizes a complex objective function assigning different weights to aspects of the solution. We found that such a combined model was not at all scalable, and therefore use a decomposition of the overall problem into multiple steps. This decomposition is based on the current business practice, and is therefore easily explained and justified to end users of the system. It also allows different stakeholders to retain control of their own process elements, which also improves user acceptance of the tool. The four phases of the planning tool are shown in Figure 1.

Clustering Visits. In the first phase, we group remote locations that are in close proximity into clusters that should be visited together. This avoids considering sub-optimal solutions that require too much travel in rural areas. The clusters will be used as input components in the next step.

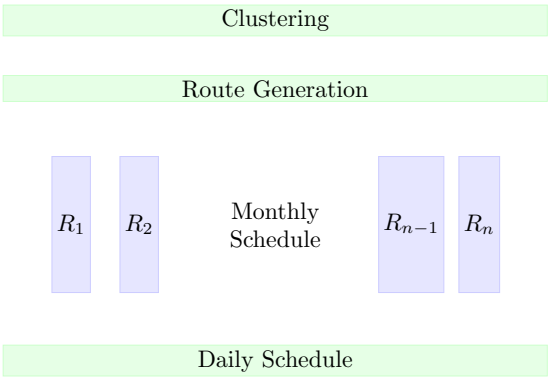


Fig. 1. Overall Problem Decomposition

Route Generation. A key requirement for the solution is that the same technician should be visiting a location over a longer period of time, as familiarity with the location and customer improves service quality. In the second phase we therefore partition all sites into assigned subsets for each technician. These sets are called routes in the industry, although at this point there is no assigned sequence order of the visits. The key objective here is to assign balanced workloads for all technicians within a depot or a region, while respecting constraints on the skill set and preferences of the technicians themselves. At the same time, the routes should be compact, so that travel between locations can be minimized in later stages of the process.

Monthly Schedule. Once we know which customer sites are assigned to which technician, we can schedule the monthly work plan for each technician independently, decomposing the problem even further. For each month, we consider all mandatory work, tasks that need to be performed during the month, as well as optional work, tasks that may be performed, if time is available. Each task has a due date, computed from the last visit to the location and the service interval. We want to avoid visiting the site too early, as this increases the overall workload, but also avoid a late visit, as this may incur penalties, and in general reduces customer satisfaction. Given the set of tasks, we have to come up with a tour for each day, visiting some customer sites in sequence, while respecting working time rules, and minimizing unproductive travel between sites.

Daily Rescheduling. The initial monthly plan will rarely be implemented "as is". On a given day, unplanned tasks, caused by equipment failure, may take precedence over initially planned work. Some tasks of previous days may be unfinished, or may require a return visit as required spare parts have just arrived. The daily rescheduling considers all these unplanned activities together with the initial plan, and comes up with a revised schedule. In order to achieve the promised response time for repair tasks, we may have

to reassign work between technicians, and use the technician nearest to a fault location, even if the customer is not in their assigned route. The details of this scheduler are not described in this paper, as the model is a variant of the monthly scheduler.

In the following sections, we will describe the details of the different stages of the tool.

3. Clustering

We use the clustering stage to group together customer sites that are in close proximity to each other. In order to reduce overall travel, planned visits to these sites should be performed together. When estimating the total amount of travel, we therefore only want to count the distance to the cluster (plus some intra-cluster travel), not the distances to each site individually. This is true only for planned visits; multiple simultaneous unplanned visits to a single cluster happen rarely, and so we must count the distance of each unplanned trip individually.

3.1. *Connected Component Based*

Our first clustering method uses a simple graph-based representation. We generate an undirected graph consisting of one node for each customer location. Nodes are connected if either

- they are located in the same city
- or they are less than d km apart
- or the connection has been manually specified by the user

We use a standard algorithm for finding the connected components in the graph, and group the sites based on their component number. The parameter d controls which items should be grouped in the same cluster.

3.2. *Hierarchical Agglomerative Based Clustering*

In this second clustering method, initially, each site represents a cluster, then, the two closest clusters are repeatedly combined until no more merging is possible. At each iteration, a cluster is represented by its clustroid, by which we mean the site with the smallest maximal path distance to the other sites in the cluster. The merge of two clusters is possible if the diameter, i.e., the measure between the two most distant sites, is less than d.

To conclude, compared to the hierarchical clustering approach, in the graph-based clustering approach, any site within a cluster is at most distant of d km from at least one other site in the cluster. In this case, the diameter of a cluster may exceed d. On the other hand, in the hierarchical clustering approach, any site within a cluster is at most d km distant from all other sites in the cluster. The

diameter of the cluster is contained within d, but some sites that are less than d apart may be in different clusters. As a result, for a given limit d, the graph-based clustering approach may produce fewer clusters of larger size while the hierarchical clustering may produce more clusters of smaller size.

4. Route Generation

We use the generated clusters of the previous stage as input for the route generation. This procedure partitions the clusters into sets (called routes by the end users), while balancing the expected yearly workload for each technician. This can be seen as a variant of bin packing, where we have to determine the best overall size of a fixed number of bins (representing the technicians) to hold the required work for all customer sites. We include both planned maintenance working time, and an estimate of unproductive travelling time in the overall workload, and also include an estimate of the unplanned repair work that will be required. As this bin packing problem consists of placing many small items (compared to the bin sizes) into the bins while minimizing the largest size used, available methods of Constraint Programming [25,24,21] will not propagate enough, and instead we use off-the-shelf Integer Programming solvers for solving the models.

We describe four variants of the model, including or ignoring some of the underlying constraints.

- The core model (which we describe in Section 4.1) deals with each customer site individually, resulting in a large model that only handles the main constraints and preference requirements.
- The second model, called the virtual route centre mode (described in Section 4.2), adds the distance between sites within the same cluster, and the objective is a combination of a balancing the workload between the agents and minimizing the intra-route travel times. The method introduces the concept of a virtual route centre to limit the travel between customer locations in a route. This is achieved with an iterative model, which repeatedly solves Integer Programming models until the route centres stabilize. We can improve the quality of the solutions by running the methods repeatedly with different start values, to avoid local minima.
- In the next model, called aggregated route generation (described in Section 5), we work directly with the clusters generated in the Clustering stage. A cluster may be fully allocated to an agent, or may be split proportionally between routes. In the latter case, the detailed assignment of the customer sites to routes is performed in yet another sub-model.
- The last model considered is a local search routine, described in Section 6. In this model, clusters and/or customers sites are moved between routes to improve the workload balance. We study the effect of selecting subsets of the available movement rules.

4.1. *Core Model*

We consider a problem with a set of sites B, which should be assigned to routes R from a set of depots D. The set of components C forms a partition of B. Lower case symbols b, c, d, r denote members of these sets. In addition, we use the following constants:

t_{ab} travel time between two locations a and b
h_r home location for route r
$l_{b/c/d}$ location of site b, component c, or depot d
$v_{b/c}$ yearly planned visits to site b or component c; $v_c = \max_{b \in C} v_b$
u_b yearly unplanned visits to site b, using a forecast based on historical data
n_d number of routes allocated to depot d
w_{rb} yearly work for site b when performed by route r, includes travel time for unplanned, but not for planned visits
$q_b^{p/u}$ average time needed on a planned (p) or unplanned (u) visit to site b
c_b component to which site b belongs
$p_{r/b}$ administrative region to which route r or site b belongs
$d_{r/b}$ depot to which route r or site b belongs in current assignment
m_b route to which site b is assigned in current, manual assignment
s_{rb} true if route r is skilled to perform work in site b

We define the yearly work w_{rb} for site b assigned to route r as the total of all unplanned visits to the location, considering the working time and an estimate of the travel time, and the work performed in the site during all planned visits.

$$w_{rb} := v_b q_b^p + u_b(t_{h_r l_b} + q_b^u) \tag{1}$$

Note that this is an approximation, as unplanned trips may start during the day in a location different from the home location.

We allow the following optional settings to enable/disable certain constraints and preferences as follows:

respectProvince. Work in each administrative area should only be assigned to technicians that are registered to work in that administrative area. If this is not enforced, then technicians missing the accreditation must perform the required training/exams, at extra cost.

respectDepot. Work currently assigned to a depot should not be assigned to workers from outside that depot. Any changes in depot assignment needs the agreement of multiple stakeholders, and are really management decisions. It can be useful to remove this restriction to see if any changes of the assignment would improve other cost factors, and which sites would be reassigned.

keepAssignment$_b$. Site b should stay with the current assignment m_b. This may be required by the technician or the owner/operator of the site, and would be considered only for some sites.

enforceSkills. The equipment in site b requires training that not all technicians may have. When enforcing this constraint, the current training is taken into account; if it is not enforced, the optimal solution may require additional training for some technicians, but may also allow for less travel.

We define a Boolean function `infeasible` which, based on the options and constant values, decides if site b can be assigned to route r.

$$\begin{aligned} \text{infeasible}(r,b) := \quad & p_r \neq p_b \wedge \text{respectProvince} && (2)\\ \vee\ & d_r \neq d_b \wedge \text{respectDepot} && (3)\\ \vee\ & r \neq m_b \wedge \text{keepAssignment}_b && (4)\\ \vee\ & \neg s_{rb} \wedge \text{enforceSkills} && (5) \end{aligned}$$

We use three sets of variables in our core model. The Boolean variables x_{rb} denote if site b is assigned to route r. The Boolean variables y_{rc} denote if component c is visited by route r; there may be multiple routes visiting a component. The non-negative variables z_d denote an upper bound on the work load of all routes in depot d. They are used to balance the workload between the routes of each depot.

The objective function (6) of the model consists of three parts. The first is the sum of the upper work load bounds for each depot, used to balance the workloads of the routes in the depot. We weight each term with the number of routes in the depot to give more importance to large depots. The second term considers the travel time used to visit the components for regular (scheduled) service. As we plan to perform all work in a component at the same time, we only have to consider this travel once for the component. The third cost element is the sum of the assigned workload for each route, which consists of the working at the site (both planned and unplanned), and the travel time required for unplanned visits.

Note that all cost terms are expressed as time durations, making them directly comparable.

$$\min \sum_{d\in D} z_d n_d + \sum_{r\in R}\sum_{c\in C} y_{rc} t_{h_r l_c} v_c + \sum_{r\in r}\sum_{b\in B} x_{rb} w_{rb} \tag{6}$$

$$\forall_{r\in R}\forall_{b\in B}: \quad x_{rb} \in \{0,1\} \tag{7}$$

$$\forall_{r\in R}\forall_{c\in C}: \quad y_{rc} \in \{0,1\} \tag{8}$$

$$\forall_{d\in D}: \quad z_d \geq 0 \tag{9}$$

$$\forall_{r\in R}\forall_{b\in B}: \quad x_{rb} = 0 \quad \text{if} \quad \text{infeasible}(r,b) \tag{10}$$

$$\forall_{r\in R}\forall_{b\in B}: \quad x_{rb} \leq y_{rc_b} \tag{11}$$

$$\forall_{r\in R}\forall_{c\in C}: \quad y_{rc} \leq \sum_{b\in c} x_{rb} \tag{12}$$

$$\forall_{b\in B}: \quad \sum_{r\in R} x_{rb} = 1 \tag{13}$$

$$\forall_{r\in R}: \quad \sum_{b\in B} x_{rb} w_{rb} + \sum_{c\in C} y_{rc} t_{h_r l_c} v_c \leq z_{d_r} \tag{14}$$

We cannot assign a site to a route if that assignment is infeasible given the current options (Eq 10). If a site in a component is assigned to a route, the route must visit the component (Eq 11). On the other hand, a route does not visit a component, if none of the sites inside the component are visited (Eq 12). Each site must be assigned to exactly one route (Eq 13). The work load assigned to each route in a depot is limited by the upper workload bound for the depot (Eq 14).

4.2. *Virtual route centres, minimizing intra route travel*

A key issue in the core model is that required travel between customer sites is not considered when making the assignment choices; the model only considers the travel from and to the home location. If an agent visits multiple customer locations on the same day, the travel required may be a significant part of the total daily working time. It is hard to include these travel times directly into the model, without solving the scheduling problem at the same time. We therefore use an approximation of the constraint, trying to reduce the diameter of the assignment route area. For this we use the concept of a virtual route centre, while adding another cost term, which is the distance between the route centre and the customer location. Reducing this cost should also lead to a reduction of the travel times between customer sites. As the assignment changes, we have to recompute the virtual route centres, leading to an iterative algorithm, where we repeatedly solve the assignment problem and recompute the route centres, until the centre locations stabilize. We introduce the notation for the virtual route centre location of route r at iteration k, denoted by l_r^k. The initial assignment with index 0 is either a random assignment, or is chosen from an existing route assignment.

We add a new term to the objective function (6), which minimizes the distances from the current virtual route centres to the assigned sites.

$$\min \sum_{d \in D} z_d n_d + \sum_{r \in R} \sum_{c \in C} y_{rc} t_{h_r l_c} v_c + \sum_{r \in R} \sum_{b \in B} x_{rb} (w_{rb} + t_{l_r^k l_b} v_b) \tag{15}$$

The overall solver is described by the procedure `findAssignment` in Figure 2, which returns true if a solution is found. The notation x^* is used to refer to the value of variable x in the optimal solution of the MIP model. The MIP model itself consists of the objective (15) together with constraints (7) to (14). We determined the values for the parameters *limit=20* and *cutoff=0.05* empirically.

```
boolean findAssignment(){
  status := true; k := 0; for(r:R) l_r^k := (Σ_{b∈B:m_b=r} l_b) / |{b∈B:m_b=r}|;
  do {
    k++;
    status &= solveMIP();
    for(r:R) l_r^k := (Σ_{b∈B} x*_rb l_b) / (Σ_{b∈B} x*_rb); % recomputeCenters(k);
  } while (k < limit && Σ_{r∈R} |l_r^k − l_r^{k−1}| > cutoff);
  return status;
}
```

Fig. 2. Pseudocode for virtual route centre optimization

5. Aggregated Route Generation

The model in Section 4.2 solves a medium sized MIP problem at each iteration of the algorithm. This can be a performance problem, especially if we want to run the solver interactively, and therefore expect results in seconds, and not minutes. We can reduce the size of the model, and therefore improve the expected runtime, if we introduce the the decomposition shown in Figure 3, using the results of the clustering method described in Section 3. We compute an initial partial assignment based on the components obtained by the clustering, this decides which components are (partially) visited in which route, and which fraction of the workload is performed by which route. In a second stage, we complete the assignment by checking each component in turn. If the component is completely assigned to one route, all buildings in the component are directly assigned to the route. If multiple routes are each assigned a fraction of the workload, then we need to run another solver (shown in red in Figure 3) to assign the correct fraction of the workload to each route. We will need this second solver in particular, if the overall workload in the component exceeds the capacity of a single agent.

5.1. *Partial Component Allocation*

In the first stage, we assign which fraction of each component workload is handled by each route. On one hand, we want to avoid that a route only handles a small part

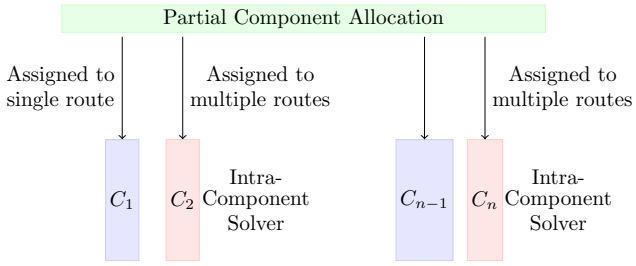


Fig. 3. Aggregated Route Generation Model, Overall Design

of the workload in a component, as we would then require to travel to the cluster area while performing only limited productive work. On the other hand, we cannot just require that each component is only handled by a single agent. This may be infeasible if the component workload is too large for a balanced solution, and may have an adverse effect on the overall workload balance between agents.

We need a few more symbols for constants for the partial component assignment as follows:

f_c smallest fractional assignment of component c
$w_{c/b}$ yearly workload for site b or component c; $w_c = \sum_{b \in c} w_b$

We define the workload for a component as the sum of the workloads of its sites, and redefine the workload of a site as the sum of the planned and unplanned work, plus the unplanned travel (using the depot centre as starting point). This slightly redefines the workload from the w_{rb} definition used in Section 4.1 to remove dependency on the assigned route.

$$w_b := u_b(t_{l_d l_b} + q_b^u) + v_b q_b^p \tag{16}$$

If we want to keep some site assigned to a specific route, we have to make sure that the component is assigned to the route. For this we introduce a function `assigned`:

$$\begin{aligned} \text{assigned}(r, c) := \exists b \in B \quad \text{s.t.} \\ c_b = c \wedge m_b = r \wedge \text{keepAssignment}_b \end{aligned} \tag{17}$$

The overall solver uses virtual route centres, and the overall iterative process defined in Figure 2. We now define the MIP model that is solved at each iteration k.

The objective function in our aggregated model is the sum of the workload bounds for each depot, the assigned work to each route, and the distances from the

virtual route centres to the assigned components. The new model is as follows:

$$\min \sum_{d \in D} z_d n_d + \sum_{r \in R} u_r + \sum_{r \in R} \sum_{c \in C} x_{rc} t_{l_r^k l_c} \tag{18}$$

$$\forall_{r \in R} \forall_{c \in C} : \quad x_{rc} \in \{0, 1\} \tag{19}$$

$$\forall_{r \in R} \forall_{c \in C} : \quad y_{rc} \in [0, 1] \tag{20}$$

$$\forall_{r \in R} : \quad u_r \geq 0 \tag{21}$$

$$\forall_{d \in D} : \quad z_d \geq 0 \tag{22}$$

$$\forall_{r \in R} \forall_{c \in C} : \quad y_{rc} \leq x_{rc} \tag{23}$$

$$\forall_{r \in R} \forall_{c \in C} : \quad y_{rc} \geq x_{rc} f_c \tag{24}$$

$$\forall_{c \in C} : \quad \sum_{r \in R} y_{rc} = 1 \tag{25}$$

$$\forall_{r \in R} \forall_{c \in C} : \quad x_{rc} = 1 \quad \text{if} \quad \text{assigned}(r, c) \tag{26}$$

$$\forall_{r \in R} : \quad u_r = \sum_{c \in C} x_{rc} v_c t_{h_r l_c} + \sum_{c \in C} y_{rc} w_c \tag{27}$$

$$\forall_{r \in R} : \quad u_r \leq z_{d_r} \tag{28}$$

The integer variable x_{rc} encodes whether or not we visit component c as part of route r. We use fractional values y_{rc} to describe how much of the total work in component c is assigned to route r. We then introduce non-negative, continuous variables u_r to describe the work assigned to route r. The non-negative variable z_d gives an upper bound on the work load assigned to each route in depot d. If we assign some work in the component to a route, then we must visit the component. If, on the other hand, we do not visit the component, then the assigned work fraction must be zero (Eq 23). If we visit a component, then we must assign a certain minimum amount of work to the route (Eq 24). The smallest value could, for example, correspond to one site. The work in a component must be split between the assigned routes, so their percentages must add up to one (Eq 25). If we want to pre-assign a site in a component, we also have to pre-assign the component to which it belongs (Eq 26). The work load for a route consists of the regular travel time to visit all assigned components and the proportion of the workload of the visited components (Eq 27). The workload of each route is bounded by the work load limit of the depot (Eq 28). If a component is assigned to a single route, then all sites in that component are automatically assigned to the route.

5.2. *Intra-Component Solver*

To complete the assignment, we need a second solver, which splits the workload in the component proportionally between the route that are assigned to the component c. This solver is run separately for each component, limiting the problem size that we need to consider in each instance. We introduce the work f_r that should be given to route r in component c, based on the fractional value assigned in the optimal

solution to (18), and the total workload of the component:

$$f_r := y^*_{rc} w_c \tag{29}$$

The objective is to group sites in the component assigned to the same route together, again using the virtual route centres.

$$\min \sum_{r \in R} \sum_{b \in B} x_{rb} t_{l^k_r l_b} \tag{30}$$

$$\forall_{r \in R} \forall_{b \in c} : \quad x_{rb} \in \{0, 1\} \tag{31}$$

$$\forall_{b \in c} : \quad \sum_{r \in R} x_{rb} = 1 \tag{32}$$

$$\forall_{r \in R} : \quad f_r(1 - \epsilon) \leq \sum_{b \in B} x_{rb} w_b \leq f_r(1 + \epsilon) \tag{33}$$

$$\forall_{r \in R} \forall_{b \in c} : \quad x_{rb} = 0 \quad \text{if} \quad \text{infeasible}(r, b) \tag{34}$$

The decision variables x_{rb} state whether site b in component c is handled by route r. Each site must be assigned to exactly one route (Eq 32). The workload assigned to each route is determined by the fractional values obtained from the optimal solution of the MIP in Section 5.1. We allow an ϵ variation of the workload to avoid infeasible sub-problems (Eq 33). We restrict the assignment of sites to feasible routes (Eq 34). The resulting problem may be infeasible, and we either have to increase the value of ϵ, and accept a more unbalanced assignment within the cluster, or we have to go back to the component assignment model, and introduce more constraints, restricting for example which agents can be assigned to the cluster. As the assignment may change the location of the virtual route centres, we run the algorithm as part of an overall route centre loop, already shown in Figure 2, until the centre locations stabilize.

6. A Local Search Approach to Route Generation

We now propose a local search approach to the route generation problem introduced in Section 4. In the algorithm we maintain a, which is a mapping assigning the buildings in B to the routes in R, $a : B \to R$. We recall that each building is associated with a cluster and technicians are expected to carry out all planned visits in a cluster at the same time.

In what follows we describe:

- The evaluation and check of feasibility of a building assignment.
- Types of move.
- The algorithm of the approach.

6.1. *The evaluation and check of feasibility of a building assignment*

We recall that the objective involves three elements: the weighted sum of the maximum work load of the routes associated with each depot, the travel time used to

```
abstract class Move {
    Move(){ }

    abstract ResetList tryMove();

    void unTry(ResetList rl){
        for(Reset reset:rl){
            int old := reset.getOld();
            int b := reset.getB();
            change(b,a[b],old);
        }
    }

    void change(int b,int oldRoute,int newRoute){
        a[b] := newRoute;
        rc[newRoute][ccAssignment[b]]++;
        rc[oldRoute][ccAssignment[b]]--;
    }
}
```

Fig. 4. Move abstract class

visit the cluster for regular (scheduled) service and the assigned workload for each route.

We keep an array of integers rc, where rc_{rc} is the number of building in component c that are assigned to route r. Given this, we first define the work load of a route r as follows:

$$w_r(a, rc) := \sum_{c \in C} (rc_{rc} > 0) \times t_{h_r l_c} v_c + \sum_{b \in B} (a_b = r) \times w_{a_b b} \tag{35}$$

In the actual implementation we also maintain $w_r(a, rc)$ since it is easy to update it when rc and a_b are updated.

We now can define the cost of an assignment in term of w_r as follows:

$$cost(a, rc) := \sum_{d \in D} n_d \times \max_{r \in d} w_r(a, rc) + \sum_{r \in R} w_r(a, rc) \tag{36}$$

Removing a building from a route would lead to no violation, but adding a building to a route may violate some integrity constraints, so if we are adding building b to route r' we need to make sure that infeasible(r', b) holds. Some of these feasibility checks will be performed during the generation of the neighborhood as we will explain later, so that only few will need to be performed during the search itself.

6.2. *Types of Move*

We now describe the types of move that we may perform during search. In each case we indicate how we update the data structures that we are maintaining. As moves are tentative (i.e., we may not commit to a move) we need to provide a way to undo the move. In Figure 4 we show the abstract class `Move`. All the moves that we are

```
// change building b to route r
ResetList tryMove(){
    ResetList res := new ResetList();
    int old := a[b];
    if (old == r) {
        // nothing to do
        return res;
    }
    res.add(new Reset(b,old));
    change(b,old,r);
    return res;
}
```

Fig. 5. Implementation of Move 1

```
// change component c to route r
ResetList tryMove(){
    ResetList res := new ResetList();
    for(int b:buildings(c)){
        if (a[b] != r){
            res.add(new Reset(b,a[b]));
            change(b,a[b],r);
        }
    }
    return res;
}
```

Fig. 6. Implementation of Move 2

```
// exchange routes of buildings b1 and b2
ResetList tryMove(){
    ResetList res = new ResetList();
    int old1 := a[b1];
    int old2 := a[b2];
    if (old1 == old2) {
        // nothing to do
        return res;
    }
    res.add(new Reset(b1,old1));
    res.add(new Reset(b2,old2));
    change(b1,old1,old2);
    change(b2,old2,old1);
    return res;
}
```

Fig. 7. Implementation of Move 3

using extend this class. The way we undo a move and update the data structures is common to all moves, so these methods are implemented in the abstract class. The implementation of each move only involves implementing the `tryMove` method, which returns the list of old assignments that need to be considered to undo the move. In the implementation of the moves `res` refers to this list.

- **Type 1: Assigning a building to a route**. We assign a building b contained in a component c, previously assigned to a route r, to a route r'. We present the implementation in Figure 5. We first check whether the new

```
// exchange routes of component1 c1 and c2
ResetList tryMove(){
    ResetList res = new ResetList();
    int r1 := uniqueRoute(buildings(c1));
    int r2 := uniqueRoute(buildings(c2));
    if (r1 == -1 || r2 == -1 || r1 == r2) return res;
    for(int b1:buildings(c1)) {
        res.add(new Reset(b1, r1));
        change(b1, r1, r2);
    }
    for(int b2:buildings(c2)) {
        res.add(new Reset(b2, r2));
        change(b2, r2, r1);
    }
    return res;
}
```

Fig. 8. Implementation of Move 4

```
<int[],int>IteratedConstraintBasedLocalSearch(){
    Neighborhood neighborhood := createNeighbors();
    int[] astar := randomAssignment();
    <astar,cstar> := constraintBasedLocalSearch(astar);
    do {
        int[] a := changeAssignment(astar,getRestartFraction());
        <a,c> := constraintBasedLocalSearch(a);
        if (c <cstar){
            astar := a;
            cstar := c;
        }
    } while (!done());
    return <astar,cstar>;
}
```

Fig. 9. IteratedConstraint-BasedLocalSearch

```
private <int[],int> constraintBasedLocalSearch(int[] a) {
    int[][] rc := createRc(a);
    int c := cost(a, rc);

    while(neighbors.hasNext() && !done()){
        Move n := neighbors.next();
        ResetList rl := n.tryMove();
        if (rl.size() > 0) {
            int cTry := cost(a, rc);
            if (cTry < c) {
                neighbors := neighbors.shuffle();
                c := cTry;
            } else {
                n.unTry(rl);
            }
        }
    }

    return <a,c>;
}
```

Fig. 10. ConstraintBasedLocalSearch

state is equal to old state, in which case we return the empty list. Otherwise, we add the old assignment to `res` and make the corresponding change.

- **Type 2: Assigning all buildings of a component to a route**. We assign all buildings of a component c to a route r. We present the implementation in Figure 6. Notice that we only save the old assignment and make a change when it is needed, i.e., $a_b \neq r$.
- **Type 3: Swapping buildings between two routes**. We have a building b_1 belonging to a route r_1 and a building b_2 belonging to a route r_2. After the move we have that that b_1 belongs to r_2 and b_2 to r_1. Figure 7 shows the implementation of this move. If the buildings belong to the same route we skip the move. Otherwise, we save the old assignments and carry out the corresponding changes.
- **Type 4: Swapping components between two routes**. Let c_1 and c_2 be two components where all their buildings are previously assigned to routes r_1 and r_2 respectively. In this move, we assign all the buildings in c_1 to r_2 and all the buildings in c_2 to r_1 (see Figure 8). We check that all buildings of each component are assigned to a unique route. `uniqueRoute` returns -1 if the buildings of the given component are assigned to more than one route. Otherwise it returns the unique route to which all the buildings of the given component are assigned. If one component is assigned to multiple routes or if r_1 is equal to r_2 we skip the move. Otherwise, we proceed with the reassignment of the routes.

6.3. *Algorithm of the approach*

In Algorithm 9 we show the implementation of our iterated constraint based local search approach. The algorithm first creates a random assignment and assigns it to a. Then it computes a local minimum from a, which is assigned to a^*. Then we enter in a loop that switches from the local search phase to the perturbation phase when a local minimum is observed. In the perturbation phase we compute another random assignment keeping a fraction of the previous assignment. We maintain the best solution found so far based on the evaluation of the cost function.

In Figure 10, we present the local search approach. After initializing rc and computing an initial set of neighbors, we start the exploration of the neighbors. At each iteration we consider a move from the neighborhood and check whether a better cost has been obtained. If that is the case, we update the generator of neighbors. Otherwise, we return to the previous state (i.e., the state before the move) by undoing the move via the invocation of the `unTry` method. We keep on looping until the condition for termination is met.

We remark that the generation of the neighborhood is performed only once. The reordering of the neighborhood during search is performed lazily as a constant-time operation.

7. Monthly Scheduling

The third stage of the problem solver considers a monthly plan for each agent, and these are created independently from each other. The plan includes pre-defined, fixed activities like planned holidays or regular training sessions, together with the required planned maintenance activities from all customer sites in the agent's route. These activities have given due-dates, but can be moved in time at a cost. If the agents visits the customer site earlier than the due-date, this will pull the next visit forward, increasing the overall workload. Visiting the customer sites after the due-date decreases customer satisfaction, and may lead to penalties depending on the total delay. On the other hand, if we impose hard limits on the time-window for a customer visit, then it may be difficult to find good tours for each day, and non-productive travel time may dominate the schedule.

The agents work fixed office hours, from 08:00 to 17:00, including a one hour lunch break. In an urban setting, travel from home to the first scheduled location, and travel from the last customer site back to home is not included in the paid working time. For multi-day trips in rural areas, these activities are included as working time, and there are additional costs and allowances for overnight stays.

We solve the monthly scheduling problem by a decomposition into two steps. We first create feasible configurations for a tour, which is a sequence of visits to customer sites that fit into the allocated time and satisfy the working rules. A configuration may cover a single or multiple days. If the problem space is small enough, we can create all relevant configurations initially, rejecting tours that require too much travel or do not include enough productive work. We can also adapt column generation [9] to create configurations on demand during the second step of the solver.

The second part of the solver selects configurations to fit into the monthly work-plan, minimizing the cost for early and late visits, while maximizing productive work, and ensuring that all required activities are covered. This requires a variant of a set partitioning solver [11]. The decomposition into a configuration generator and a set partitioning selection process is a standard tool for transportation problems, well known in the literature [7,16,27]. Our solver extends this general scheme with constraints and objectives to handle earliness and lateness of visits.

7.1. *Configuration Generation*

A configuration is a potential sequence of visits to customer sites, which fits into a single day (for urban visits), or may extend to multiple days for rural areas. To check for feasibility, start and end times are associated with each visit, and travel between locations is taken into account. Figure 11 shows a typical single day configuration, where travel from and to home is not included as working time, but travel during the day between sites counts as (non-productive) working time. Time for a one hour lunch break is also included in the configuration. Figure 12 shows a typical two-day configuration, where the agent stays overnight at a hotel. The potentially long travel

from home to the first customer location and back home at the end of the trip are counted as working time.

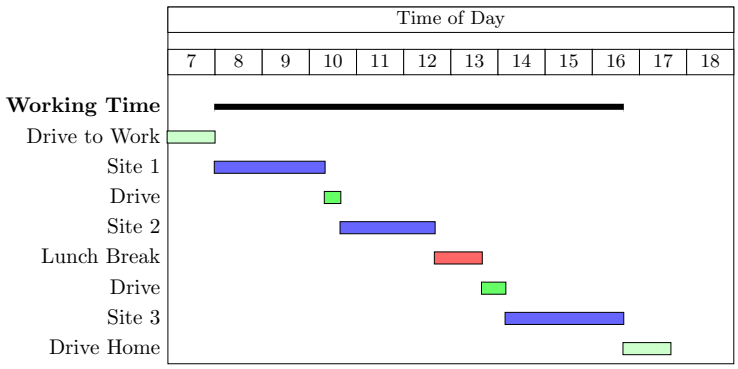
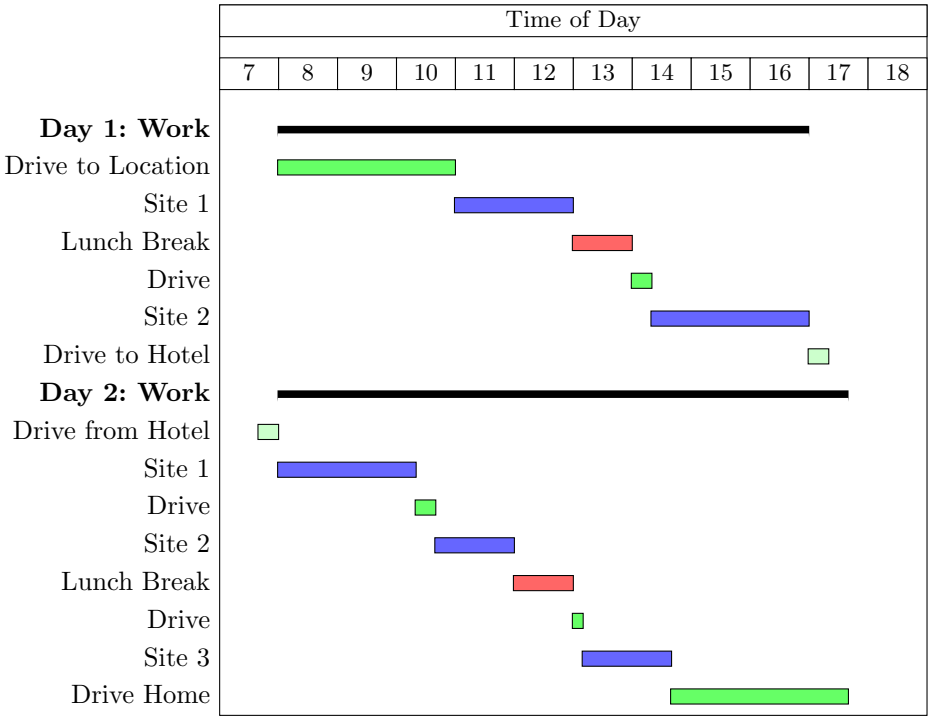


Fig. 11. Single-day Configuration (Work, Travel, Lunch Break)

Fig. 12. Multi-day Configuration (Work, Travel, Lunch Break)



We use dynamic programming [2] to systematically create feasible configurations,

starting with all configurations consisting of a single site visit only. We then, recursively, extend existing configurations by adding an additional activity to the given sequence, inserting them between given visits and replacing the travel between the sites to consider the new location. We only keep the best insertion, and can reject configurations which are too inefficient. The process stops if we can add no more work to the configuration without violating the working time limits.

Building the sequence of visits incrementally does not guarantee minimality of the required travel; we have to re-optimize each configuration as a TSP (travelling salesman problem) to achieve optimality. We currently skip that step to improve overall response time.

7.2. *Configuration Selection*

We now describe a MIP model for the configuration selection, which considers the set of all required activities A, using a set of previously generated configurations C, while planning the schedule for a set of days D, defined by the planning period and the overall calendar. We denote as set E all excluded days, where the person is already pre-assigned to other activities (training, holidays). If the schedule of the previous month finished with an incomplete multi-day trip, we can also use the set E to reserve time to finish that trip.

We require the following constants:

p_{ca} Boolean which states if configuration c contains activity a
n_a cost of not performing operation a in the planning period
q_{cd} cost of running configuration c on day d
m_c duration (in days) of configuration c, used to handle multi-day trips

$$\min \sum_{c\in C}\sum_{d\in D} v_{cd}q_{cd} + \sum_{a\in A} u_a n_a \tag{37}$$

$$\forall_{c\in C}\forall_{d\in D}: \quad v_{cd} \in \{0,1\} \tag{38}$$

$$\forall_{a\in A}: \quad u_a \in \{0,1\} \tag{39}$$

$$\forall_{a\in A}: \quad \sum_{c\in C|p_{ca}}\sum_{d\in D} v_{cd} + u_a = 1 \tag{40}$$

$$\forall_{d\in D}: \quad \sum_{c\in C}\sum_{\substack{d'\in D\\ d'\le d\\ d-d'<m_c}} v_{cd'} \le 1 \tag{41}$$

$$\forall_{c\in C}: \quad \sum_{d\in D} v_{cd} \le 1 \tag{42}$$

$$\forall_{d\in E}\forall_{c\in C}\forall_{\substack{d'\in D\\ d'\le d\\ d-d'<m_c}}: \quad v_{cd'} = 0 \tag{43}$$

We use Boolean decision variables v_{cd} to state that we schedule (or start, for multi-day trips) configuration c on day d. We may not be able to handle all activities in the given time period. Instead of just reporting an infeasible problem, we use Boolean variables u_a to postpone activities until the next time period, and highlight capacity issues to the user. The objective (37) is to find the best selection of configurations to run on each day, so that their cost (earliness and lateness), together with the penalties for non-performed activities, is minimized. Each activity must be either performed once in one configuration on one day, or must be postponed until after the current planning period (Eq 40). Finally, we state that on each day d we can run at most one configuration, allowing for a case with so little work that we can have a free day as well. To handle multi-day configurations, we consider all starting days d' for a configuration c so that the configuration is still running on day d (Eq 41). We also state that each configuration can be used at most once (Eq 42). For all excluded days d in set E, we cannot run any configuration that would be active on that day (including multi-day trips that have started, but not yet finished) (Eq 43).

8. Experiments

The end-user provided a real-world dataset for the evaluation of the algorithms, which covers a large geographical area, covering five depots in three second-level administrative divisions. This area covers 1,252 customer sites, which are visited by 18 technicians. The amount of work required per customer sites varies significantly, depending on the number of devices installed at the site, and the service level agreement with the customer. In total, we have to consider 79,454 activities during the one-year planning horizon. All experiments were run on a Windows 10 laptop with a 2.9GHz Intel i7 CPU, and 64GB of memory, using Cplex 12.63 as the MIP solver.

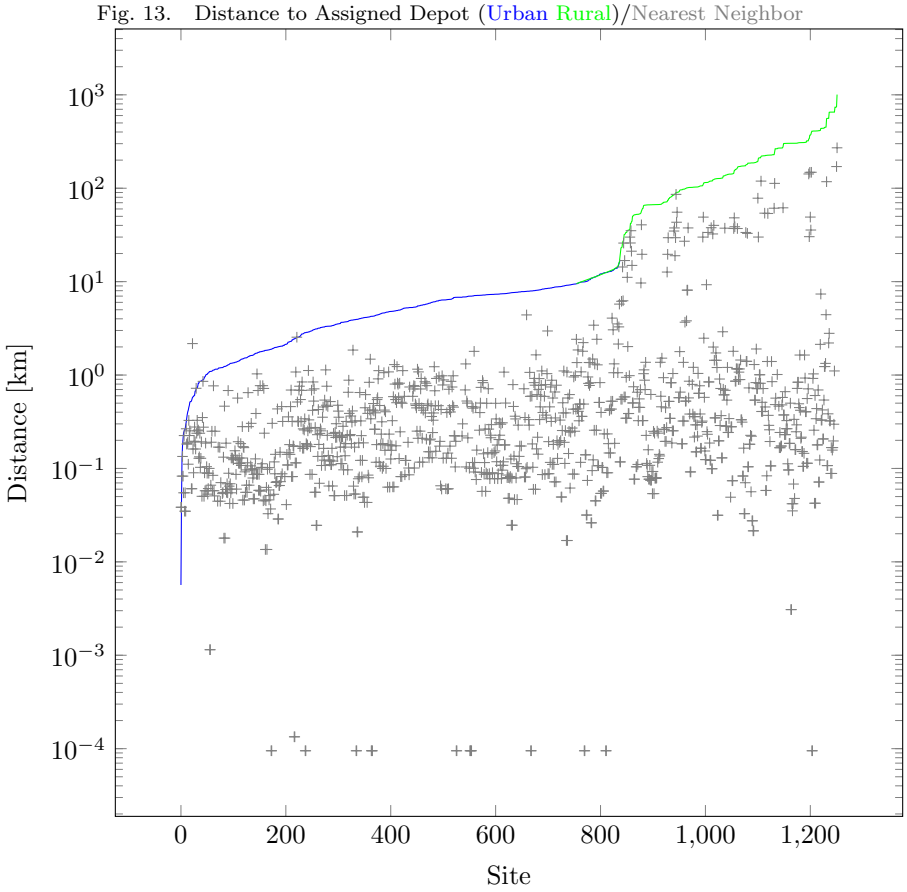
8.1. *Characteristics of real-world data set*

We use real-world data from one geographical region of operation of our industrial partner. As we can not make these data publicly available, we try to characterize the data with the following analysis.

Figure 13 plots the distance from each site to the assigned depot, sorting the sites by increasing distance. Sites in urban areas are shown in blue, sites in rural areas in green. We also plot for each site the distance to the nearest neighbour (in gray). The sites furthest away from a depot are over 1,000 km distant, but for many the nearest neighbour site, that will also be visited, is much closer, even less than 1km away. A few sites are singletons, far away from the depot, and also more than 100km away from the nearest neighbour. As depots are located in urban areas, all sites there are rather close to their depot.

Figure 14 plot the yearly workload (in hours) of each site against the distance from the depot. The sites with the highest workload (over 100 hours per year) are

Fig. 13. Distance to Assigned Depot (Urban Rural)/Nearest Neighbor



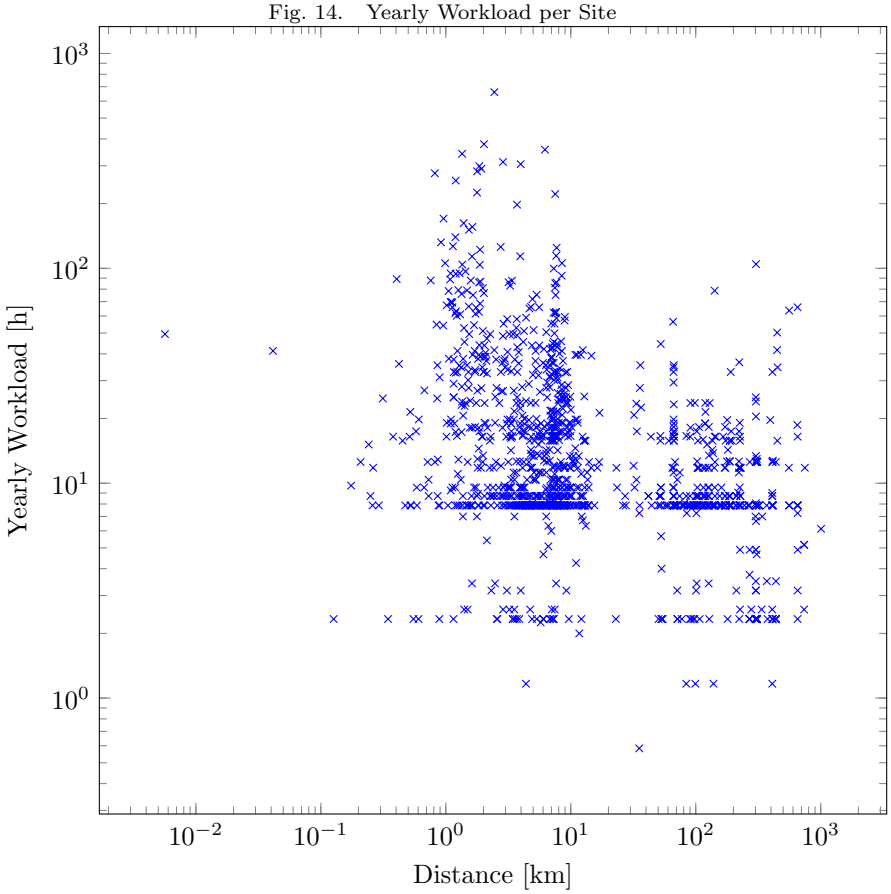
a few kilometres from the depot, but some sites far from the depot also have quite large workloads.

8.2. *Clustering*

We plot the number of clusters found as a function of distance d in Figure 15(a). As d increases, the number of clusters decreases rapidly. Enforcing the clustering together for sites in the same city (dotted) only has an impact for small values of $d < 10km$. Based on this analysis, we use a value of $d = 20km$ and the resulting nearly 100 clusters as basis for the route generation.

In the connected component based clustering, some sites may be in the same component, even though their distance is greater than distance limit d, as they are connected through a chain of other sites, each at a distance of less than d. Figure 15(b) considers the fraction of all pairs of sites that are in the same cluster, but whose distance is greater than d. We see that there is a local minimum for $d = 18km$, where less than 6% of clustered pairs exceed the distance limit. This

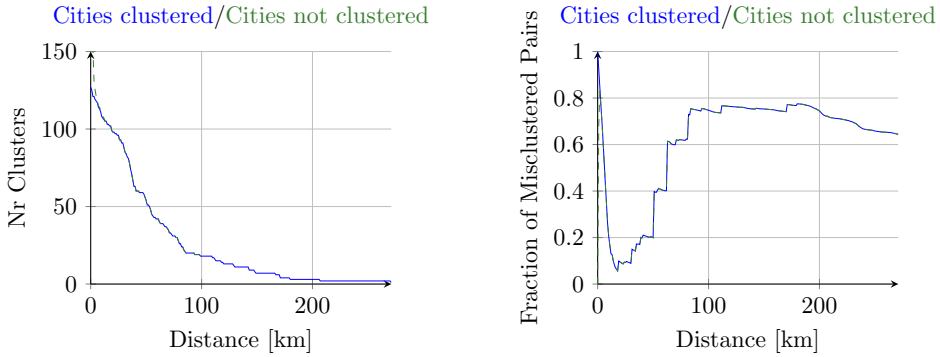
Fig. 14. Yearly Workload per Site



further justifies our choice of $d = 20km$ in our experiments.

8.3. *Comparing core model, virtual route centres, and aggregated route assignment*

Table 1 compares the three models presented in Sections 4 and 5 on the same problem instance, with two sets of constraints. In the Unconstrained Problem, we are able to re-allocate work between depots, and we ignore the required skill levels for each piece of work. In the Constrained Problem, the depot assignment is kept fixed, work can only be re-distributed between technicians of the same depot, and we enforce the skill requirement. As a use case, the unconstrained problem considers the overall problem, identifying possible changes in the current process that would lead to long-term savings, after potential reorganization of the depot areas and re-training of technicians. The constrained problem considers "low hanging fruit", changes that do not require additional investment or re-training, but that lead to



(a) Number of Clusters (b) Fraction of Misclustered Pairs

Fig. 15. Connected Component Based Clustering: Impact of Parameter d

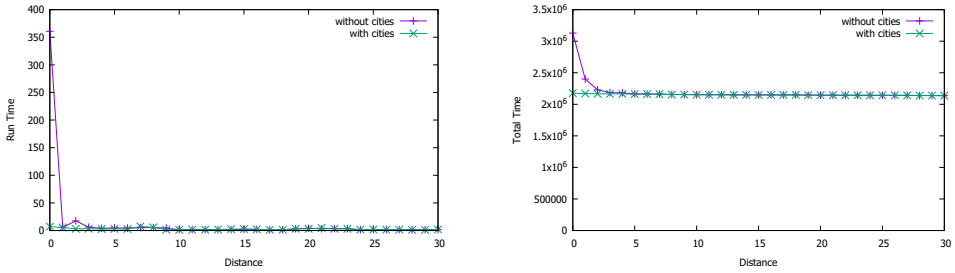
immediate savings.

In the results in Table 1 we report the amount of work scheduled, the total travel time estimated, the standard deviation of the total time allocated to each route, the average time of a trip within a route, the number of iterations of the model run, and the total run-time, marking with TO when a timeout of 300 seconds was limiting the search, and with OPT if the optimal solution was found. All three models allocate the fixed work in all customer sites, while the travel time and the balance between the route workloads is quite similar. As the core model ignores the travel within each route, its average trip duration is more than twice the value of the moving route centre model. The average trip duration of the aggregated model is slightly worse than for the moving route centres due to the independent execution of the intra-component models, but the aggregated model converges much more rapidly, while solving each sub-problem to optimality.

Table 1. Comparison of Models on Two Problem Variants

Problem	Model	Total Work (min)	Total Travel (min)	Route Std Dev (min)	Av Trip (min)	Iterations	Run-Time (s)
Unconstrained	Core	1,647,834	308,723	20199.02	188.40	-	(TO)300.00
	Moving Center	1,647,834	312,389	16717.01	67.18		77.85
	Aggregated	1,647,834	309,220	17593.68	85.72		3.67
Constrained	Core	1,647,834	334,221	5647.39	172.87	-	(OPT)44.46
	Moving Center	1,647,834	334,317	5648.01	81.28		8.21
	Aggregated	1,647,834	334,317	5690.90	100.96		5.37

For our industrial data-set, the moving route centre model is too large to run to optimality in an iterative procedure. We can either restrict run time by a timeout, and/or accept sub-optimal solutions by allowing for an optimality gap. In Table 2 we show the impact of choosing different values for the allowed gap. We indicate the run-time required, the objective value reached, the expected distance travelled



(a) Run Time (b) Solution Quality

Fig. 16. Impact of Clustering Distance (X-axis) on Results of the Aggregated Route Generation Algorithm

for the route assignment, the number of iteration steps needed to converge to a local optimum, and the number of timeouts in the solver runs. We see that while a higher gap allows for faster solver runs, we need more iterations to converge to a solution. Removing the gap option completely results in a significant increase in time required, as now each run hits the timeout limit of 300s. The best solution is obtained for a gap of 0.1%, where we need 6 iterations to converge, while only the first run times out.

Table 2. Impact of Optimality Gap on Moving Route Center Algorithm (Solver Cplex, Timeout = 300s, Cutoff =0.05)

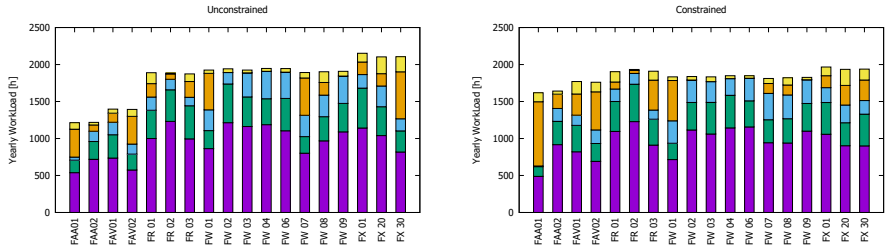
Gap	RunTime [s]	Objective	Distance [km]	Iterations	Timeout
2%	1,001.60	4,534,324	206,391	136	0
1%	744.08	4,501,536	204,266	60	0
0.5%	736.00	4,495,726	215,692	33	0
0.25%	610.38	4,498,607	206,758	18	0
0.1%	575.35	4,496,848	198,730	6	1
n/a	1,658.00	4,496,295	198,783	5	5

Figure 16 considers the impact of varying the clustering distance between 1 and 30km on the aggregated route generation algorithm, (a) shows the run-time, (b) shows solution quality. We again distinguish the case where we also automatically cluster cities (shown in blue, with x markers) or where we only use the distance to clustering locations (red, with + markers). We see that both run time and solution quality vary only very little with the choice of parameter, with the exception of values smaller than 5km, where, without clustering cities, many small clusters lead to larger problems to solve, while over-estimating the travel times. Similar results are obtained when using other clustering methods.

Figure 17 summarizes the results of the unconstrained (left) and constrained (right) route allocation. Each bar represents one technician, with the first characters of the name indicating the depot. The bar shows the elements of assigned work, from

bottom to top: preventive maintenance, planned testing, planned travel, predicted unscheduled work, predicted unscheduled travel. We can see that the total work for each technician is balanced between workers in the same depot, but not always between workers at different depots. The composition of work for each technicians varies significantly, depending on preferences and skills for each technician.

Fig. 17. Bar Chart of Allocated Yearly Workload for Routes; Unconstrained problem left, Constrained problem right



8.4. *Local Search for Route Generation*

We now present some results for the local search algorithm presented in Section 6. Figure 18 evaluates the choice of the moves that we consider for solving the problem. We choose a subset of the four move types, run 20 experiments with different random seeds for twenty seconds and show the box plot of the outcomes to evaluate the obtained solution quality. We modify one percent of the assignment at each restart of the algorithm. We see that only using moves of type one (assigning locations to a route), or using move types one or three (swapping the assignment of locations) does not lead to high quality solutions. The best results are obtained combining move type two (assigning a cluster to a route) with other moves. Including move type three (swapping assignments of locations) leads to slightly worse outcomes. This seems due to the high number of potential swaps, and the relatively low number of improving moves of this type.

To better understand the results, we also show a table of the paired t-test p-values (Table 3) comparing all considered problem variants. Cells coloured red indicate highly significant changes (better than 0.001) between the problem variants, cells coloured in yellow indicate significant (better than 0.05) changes. We see that based on the given experiment, we can not distinguish the results of using move types one and two, and move types one, two and four.

We now study the impact of the restart percentage of the solution quality. For selected moves one, two and four we vary the restart percentage between 0.5% and 5%. The results show that using smaller restart percentages usually leads to better results. If we modify the currently best solution too much, the cost of the perturbed assignment will increase significantly, and we will need more time to make that

Fig. 18. Selection of Moves

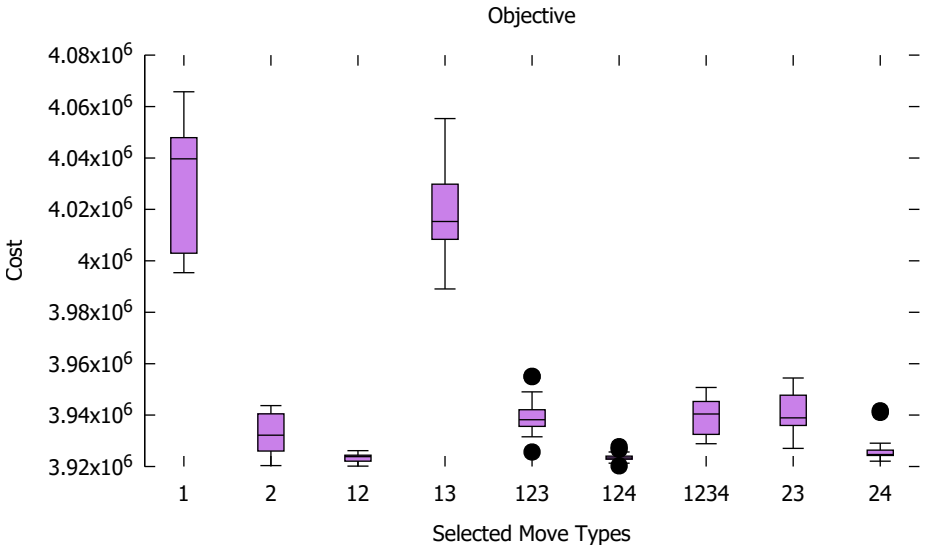


Table 3. Paired t-test p-values Comparing the Moves Selection; Colored cells indicate that differences are statistically significant

Moves	1	2	12	13	123	124	1234	23	24
1	NaN	0.0000	0.0000	0.0968	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	NaN	0.0001	0.0000	0.0037	0.0000	0.0104	0.0043	0.0129
12	0.0000	0.0001	NaN	0.0000	0.0000	0.6012	0.0000	0.0000	0.0106
13	0.0968	0.0000	0.0000	NaN	0.0000	0.0000	0.0000	0.0000	0.0000
123	0.0000	0.0037	0.0000	0.0000	NaN	0.0000	0.8484	0.7099	0.0000
124	0.0000	0.0000	0.6012	0.0000	0.0000	NaN	0.0000	0.0000	0.0207
1234	0.0000	0.0104	0.0000	0.0000	0.8484	0.0000	NaN	0.5787	0.0000
23	0.0000	0.0043	0.0000	0.0000	0.7099	0.0000	0.5787	NaN	0.0000
24	0.0000	0.0129	0.0106	0.0000	0.0000	0.0207	0.0000	0.0000	NaN

solution competitive again.

This effect is highlighted in Figure 20, which shows the evolution of the cost function during the first two seconds for selected moves two only, and two and four. The spikes in the cost function are caused by the restarts degrading the best solution found, but over time these restart changes lead to better overall solutions. Choosing a restart percentage which is too small will run the risk of not escaping a local minimum.

Figure 21 shows the solution quality of the best method variants over time for one problem instance. We can see that only using move types 2 takes more time to improve the cost, while never quite catching up with the three better alternatives, which converge quite quickly to nearly identical results. Extending the run-time does not lead to much further improvement.

Fig. 19. Impact of Change Parameter for Selected Moves 1, 2, and 4

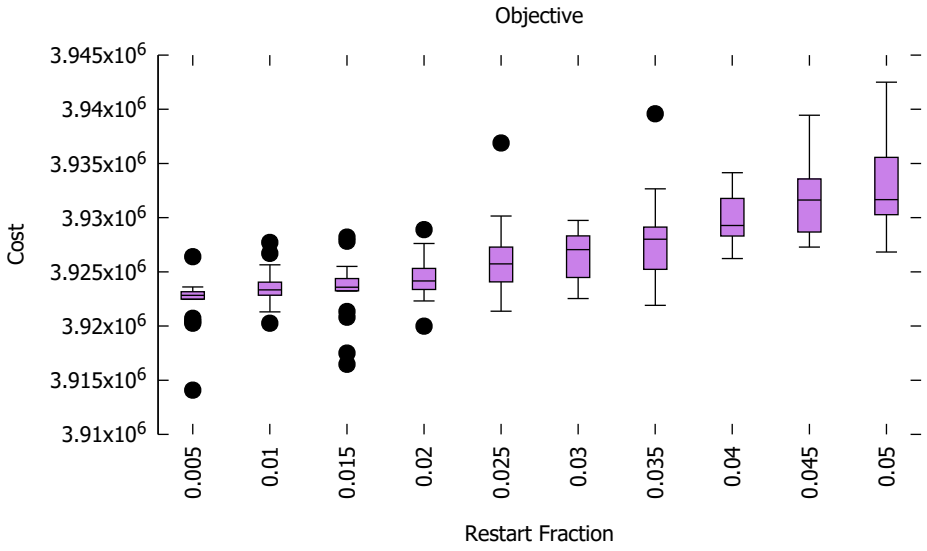
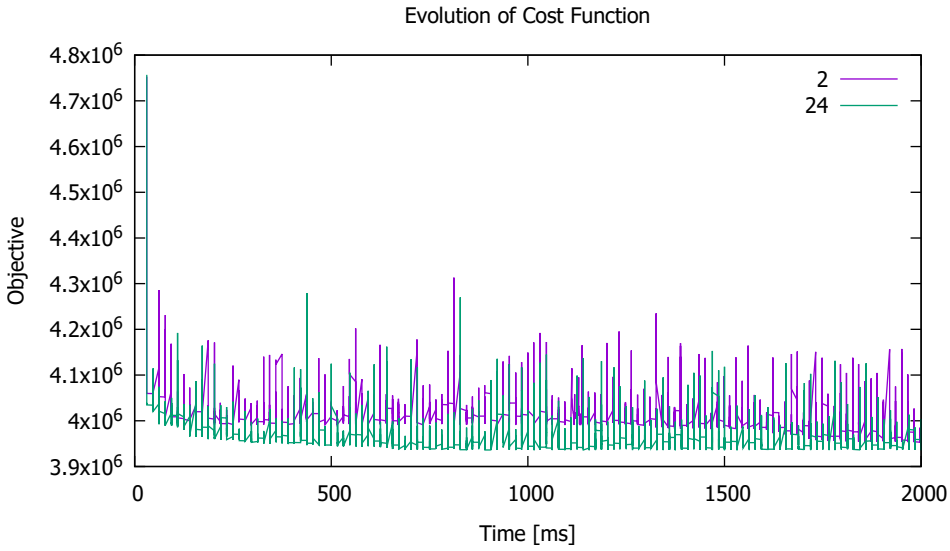


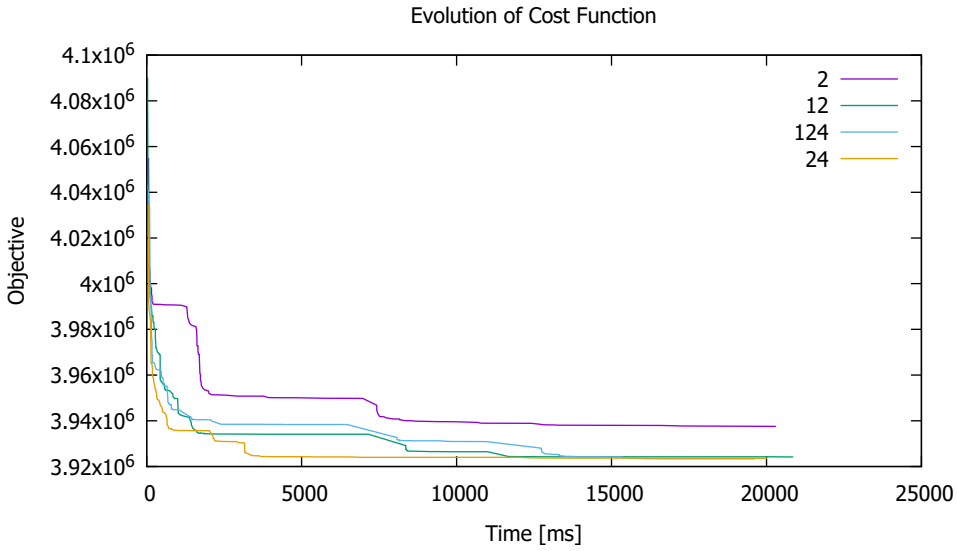
Fig. 20. Changes of Cost Function During First Two Seconds



8.5. *Scheduling*

To evaluate the Scheduling Model, we use the results of the Aggregated Route Generation model as input data, and vary the time window in which a task can be scheduled before and after its due-date. As the time window increases, more and

Fig. 21. Best Solution Found over Time



more tasks must be considered together in a configuration, therefore the number of configurations increases rapidly with the parameter value. Table 4 shows results for different time window sizes. We list the run-time (in seconds), the value of the objective function, the distance travelled for all routes, and the number of configurations generated.

Table 4. Schedule Results for One Month Based on Time Window Parameter

Time Window	Run-Time (s)	Cost	Travel (km)	Configurations
4	62.38	100,006.17	25,621.99	71,050
5	77.68	98,429.33	24,724.10	97,913
6	172.73	94,463.00	22,365.61	194,697
7	505.93	91,480.83	20,568.48	562,268
8	2,273.85	89,372.17	19,299.16	1,192,461

The results indicate that widening the time window results in a large increase of configurations to be considered, and a significant increase in the computation time. At the same time, the cost of the solution decreases by 10%, largely due to a reduction of the total travel distance needed. This outcome depends on the relative cost attached to earliness and lateness of activities, compared to the operational cost of running the schedule. Finding the best schedule therefore depends on the business objectives of the end user, and relative weights assigned to the different cost elements.

9. Conclusions

This paper describes an industrial assignment and scheduling problem for travelling repair technicians, that we have solved by a decomposition into sub-problems following the current business practice. We use different techniques for the various solution steps, and compare alternative models and approaches for the key components. Travel times between sites are computed from OSM route data, while required working times are obtained from historial experience. The solution methods used include clustering, mixed integer programming, dedicated local search methods, dynamic programming and special purpose set partitioning algorithms. In the future, we will extend this work by considering more stochastic data for travel and work times, and we will integrate the scheduling tool with a simulation platform to consider multiple scenarios for each day. We are also interested in including more preferences of the various stakeholders.

References

1. M. Antunes, V. Armant, K. N. Brown, D. A. Desmond, G. Escamocher, A. George, D. Grimes, M. O'Keeffe, Y. Lin, B. O'Sullivan, C. Ozturk, L. Quesada, M. Siala, H. Simonis, and N. Wilson. Assigning and scheduling service visits in a mixed urban/rural setting. In L. H. Tsoukalas, É. Grégoire, and M. Alamaniotis, editors, *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece*, pages 114–121. IEEE, 2018.
2. R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 11 1954.
3. F. Blakeley, B. Argüello, B. Cao, W. Hall, and J. Knolmajer. Optimizing periodic maintenance operations for Schindler elevator corporation. *Interfaces*, 33(1):67–79, 2003.
4. N. Bostel, P. Dejax, P. Guez, and F. Tricoire. Multiperiod planning and routing on a rolling horizon for field force optimization logistics. In B. Golden, S. Raghavan, and E. A. Wasil, editors, *The vehicle routing problem: latest advances and new challenges*, pages 503–525. Springer, New York, 2008.
5. J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239(1):39–67, 2016.
6. J.-F. Cordeau, G. Laporte, F. Pasin, and S. Ropke. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409, 2010.
7. F. H. Cullen, J. J. Jarvis, and H. D. Ratliff. Set partitioning based heuristics for interactive routing. *Networks*, 11(2):125–143, 1981.
8. J. V. den Bergh, J. Belin, P. D. Bruecker, E. Demeulemeester, and L. D. Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367 – 385, 2013.
9. L. Ford and D. Fulkerson. A suggested computation for maximal multicommodity network flows. *Management Science*, 5:97–101, 1958.
10. A. Froger, M. Gendreau, J. E. Mendoza, E. Pinson, and L.-M. Rousseau. Maintenance scheduling in the electricity industry: A literature review. *European Journal of Operational Research*, 251(3):695 – 706, 2016.
11. R. S. Garfinkel and G. L. Nemhauser. The set-partitioning problem: Set covering with equality constraints. *Oper. Res.*, 17(5):848–856, Oct. 1969.

12. B. Golden, S. Raghavan, and E. A. Wasil, editors. *The vehicle routing problem: latest advances and new challenges*. Springer, New York, 2008.
13. A. R. Heching and J. N. Hooker. Scheduling home hospice care with logic-based Benders decomposition. In C. Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*, volume 9676 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2016.
14. G. Hiermann, M. Prandtstetter, A. Rendl, J. Puchinger, and G. R. Raidl. Metaheuristics for solving a multimodal home-healthcare scheduling problem. *CEJOR*, 23(1):89–113, 2015.
15. C. Holland, J. Levis, R. Nuggehalli, B. Santilli, and J. Winters. UPS optimizes delivery routes. *Interfaces*, 47(1):8–23, 2017.
16. J. P. Kelly and J. Xu. A set-partitioning-based heuristic for the vehicle routing problem. *INFORMS Journal on Computing*, 11(2):161–172, 1999.
17. A. A. Kovacs, S. N. Parragh, K. F. Doerner, and R. F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, 15(5):579–600, 2012.
18. D. Lesaint, C. Voudouris, and N. Azarmi. Dynamic workforce scheduling for British Telecommunications plc. *Interfaces*, 30(1):45–56, 2000.
19. D. Lesaint, C. Voudouris, N. Azarmi, and B. Laithwaite. Dynamic workforce management. In *IEE Colloquium on AI for Network Management Systems*, pages 1/1–1/5, Apr 1997.
20. Y. Lin, A. Hsu, and R. Rajamani. A simulation model for field service with condition-based maintenance. In *Proceedings of the Winter Simulation Conference*, volume 2, pages 1885–1890 vol.2, Dec 2002.
21. F. Pelsser, P. Schaus, and J. Régin. Revisiting the cardinality reasoning for binpacking constraint. In C. Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2013.
22. S. Remde, P. Cowling, K. Dahal, and N. Colledge. Exact/heuristic hybrids using rVNS and hyperheuristics for workforce scheduling. In C. Cotta and J. van Hemert, editors, *Evolutionary Computation in Combinatorial Optimization: 7th European Conference, EvoCOP 2007, Valencia, Spain, April 11-13, 2007. Proceedings*, pages 188–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
23. A. Rendl, M. Prandtstetter, G. Hiermann, J. Puchinger, and G. R. Raidl. Hybrid heuristics for multimodal homecare scheduling. In N. Beldiceanu, N. Jussien, and E. Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*, volume 7298 of *Lecture Notes in Computer Science*, pages 339–355. Springer, 2012.
24. P. Schaus, J. Régin, R. V. Schaeren, W. Dullaert, and B. Raa. Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem. In M. Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 815–822. Springer, 2012.
25. P. Shaw. A constraint for bin packing. In M. Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.

26. S. Souyris, C. E. Cortés, F. Ordóñez, and A. Weintraub. A robust optimization approach to dispatching technicians under stochastic service times. *Optimization Letters*, 7(7):1549–1568, 2013.
27. É. D. Taillard. A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO - Operations Research*, 33(1):1–14, 1999.
28. F. Tricoire. *Vehicle and personnel routing optimization in the service sector: application to water distribution and treatment*. Theses, Université de Nantes, Feb. 2006. Thèse réalisée en convention CIFRE avec Veolia Eau.
29. S. Velicu, I. Paunescu, and M. D. Paraschiv. Transition from the classic service operations to the decision support in maintenance activity. *Proceedings in Manufacturing Systems*, 10(1):45, 2015.