universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Heuristics for Service Technician Routing and Scheduling Problems"

Verfasser

Attila Kovacs

angestrebter akademischer Grad

Magister rerum socialium oeconomicarumque

Wien, 2009

Studienkennzahl:	A 157
Studienrichtung:	Internationale Betriebswirtschaft
Betreuer:	O.Univ.Prof. Dipl.-Ing. Dr. Richard F. Hartl

Contents

1. Introduction

The everyday business for many companies that operate in sectors like telecommunications, electricity or gas supply is to create, maintain and repair their clients´ infrastructure by executing on-demand interventions. Given the new developments in the above mentioned industries, there is a steady increase in the demand for interventions and as a result, an increase in the number of competing companies as well. So, in order to save labor costs, the firms beware themselves from increasing their staff in big scales. Nevertheless, the demanded tasks must be performed anyhow, which calls for smart scheduling methods.

Because of this and the need to maintain competitiveness, technicians and interventions must be scheduled in a sophisticated manner so that as many interventions as possible can be performed by the existent pool of technicians. This economization can, in some cases, lead to unscheduled tasks, but because interventions are services to the customers that should be executed as soon as possible, these unscheduled tasks must be outsourced to external companies. The schedules that indicate who is assigned to which intervention and the sequence in which interventions must be executed are arranged by supervisors in many companies (Dutot et al., 2006). Therefore, without computational assistance the plans might demonstrate weaknesses, especially when the number of technicians and interventions is high.

The aim of this thesis is to create a fast heuristic, which is able to support the supervisors by providing schedules that are as efficient as possible. In our context, efficiency is measured by the objective function, composed of the routing costs and outsourcing costs for those tasks that cannot be scheduled.

For a solution method that is applied to such real life problems it is important to cover a large number of parameters like the specialization of the technicians in different areas with different skill levels, the geographical distribution of the demanded interventions and derived from that, the travel costs among each other. What´s more, time windows in which technicians must arrive at the intervention locations must be considered. Beyond that, some interventions might have very high skill requirements so the grouping of some technicians to a team must also be taken into account.

A heuristic approach is presented in the following chapters, which creates a schedule from scratch by considering the above mentioned requirements. The framework used

for this solution method is called *adaptive large neighborhood search* as applied by Ropke and Pisinger (2006b), Pisinger and Ropke (2007) and Cordeau et al. (2008).

2. Problem formulation

Basically, the problem denoted as *service technician routing and scheduling problem* (STRSP) can be interpreted as an extended *vehicle routing problem with time windows* (VRPTW) where a set of customers (tasks) are spatially distributed and each of them requires a determined time period to be serviced. A set of vehicles, initially located at their depot, which can have same or different characteristics are available to visit each customer and to execute the required services. These services can be a pickup or a delivery in the simplest cases and in more complex cases both, just like for the *pickup and delivery problems with time windows* (PDPTW). Furthermore, some customers are characterized by time windows defined as the earliest possible time to begin the service and the last time when servicing could still be started. This implies that if a vehicle arrives at a customer too early it has to wait until the time window of the customer begins to be able to start the service. In other words, the vehicles can have idle times. The tours created by visiting customers are associated with some costs, which are derived either from the travel distances or the travel times between each of the customers and the depot.

Moreover, problems can be distinguished on the base of the number of depots. In contrast to the *multi-depot vehicle routing problem* (MDVRP), all vehicles are located at the same depot in the VRPTW from which they start their routes and where they must return to after finishing.

In our context, the VRPTW is extended by referring to the *technicians and interventions scheduling for telecommunications problem* described in Dutot et al. (2006) and the following paper by Cordeau et al. (2008).

In the following, we denote the modified customers as tasks and the vehicles are converted in teams of technicians with different skills. The tasks require only the skills of the technicians, so no goods are transported.

Technicians are specialized in different areas with different proficiency levels to be able to work on a high variety of tasks. Tasks are independent from each other and are characterized by their skill requirement matrix requested from the technicians. Their

demand on the technicians´ skills determines the configuration of the teams. To come up with the required abilities, some technicians are grouped together into teams, which must stay together for one day. As already mentioned for the VRPTW, tasks are requested by the clients who are located at different sites and as a result, travel times and travel distances between the locations must be taken into account. Furthermore, some clients require a service within a specified time window in which the servicing of the job must be started. The execution of the interventions is urgent since this solution framework deals with the handling of service demands. So if the available set of technicians is not able to complete all of them, outsourcing can be chosen to satisfy the clients, which of course comes along with some costs.

The solution method, which is characterized by the necessity of grouping technicians, can be simplified by assuming that tasks still have different skill requirements but they can be performed by only one technician. If so, the grouping of the technicians becomes needless and one technician represents one team. Nevertheless, tasks must be assigned in a way such that a team with a single technician is able to meet all requirements.
This variation shows similarities to a more complex version of the VRPTW with a heterogeneous vehicle fleet. Such problems occur for example, when some vehicles cannot visit some customers because of their dimensions.

The requirements to perform a task can be illustrated by a p x q skill requirement matrix ($w^i_{\alpha\beta}$), where the number of technicians needed, skilled in domain p with the level q, is posted. The columns of the matrix refer to the different domains and the rows to the proficiency levels.

$$\begin{pmatrix} 1 & 2 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \end{pmatrix}$$

The values in the upper example (Cordeau et al., 2008) denote that for task i one technician is needed, which is trained in domain 1 with the proficiency level of 1. It requires 1 trained technician in domain 2 with level 3 and due to the fact that a better skilled technician can also perform simpler functions, the same technician is used for the first and the second level. Nevertheless, in this example, a second technician is needed with at least level 1 proficiency in domain 2. No knowledge is needed in domain 3 and two technicians with skill level 3 are sufficient in the fourth domain, because they are also able to execute the simpler functions.

The technicians are characterized by their skills, which again can be demonstrated in a skill matrix ($v^j_{\alpha\beta}$) where the columns of the matrix refer to the domains and the rows to the levels:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The skill matrix in the example above shows that technician j is skilled in all domains but at different levels. In domain 1 he can perform level 1 and 2 functions, in domain 2 and 3 he is only skilled at the competence level 1 and in domain 4 he is an expert and can perform all functions.

Because the upper technician doesn't have the skills to execute the upper task he must be grouped together with another technician.

Technician 1		Technician 2		Teamed together
$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	+	$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$	=	$\begin{pmatrix} 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 \\ 0 & 1 & 0 & 2 \end{pmatrix}$

The generated team would be able to carry out the task because all required skills are covered. Nevertheless, there is a waste of capacity, but this is allowed in this scenario.
Only one day is planned at once with the objective to find a schedule, which ensures minimal travel costs and in which as many jobs as possible can be assigned to the available technicians to avoid the need for outsourcing for some additional costs.

3. Mathematical model

The problem is described precisely by the mathematical model demonstrated below and is based on Xu and Chiu (2001) and Cordeau et al. (2008).
A set of tasks $N = \{1, \ldots, n\}$ has to be performed by the set of teams $\tau = \{1, \ldots, \mu\}$ whereas each team is arranged by the set of technicians $T = \{1, \ldots, m\}$.
Moreover, the model uses binary variables x_{jr} equal to one if technician $j \in T$ is assigned to team $r \in \tau$ and variables y_{ir} equal to one if task $i \in N$ is performed by team r. Binary variables z_i take value one if it is not possible to assign a task to a team and therefore it has to be outsourced.

Finally, the binary variables $u_{ii'r}$ signify whether job i´ is performed immediately after job i by team r which then implies the travel costs $c_{ii'}$. The travel costs can either be derived from the travel times $t_{ii'}$ or the travel distances $d_{ii'}$ between each task.
The tasks are characterized by their requirement matrix $w^i_{\alpha\beta}$, the earliest possible starting time e_i and the last possible ending time l_i, the duration of the service s_i, the actual starting time b_i and outsourcing cost o_i. An upper bound for the technicians to return to the depot is given by the time window end of the depot l_0. The teams can leave the depot at time zero ($e_0 = 0$).

The skills of the available technicians are displayed in the matrix $v^j_{\alpha\beta}$. Variables M denote a large constant number.

$$minimize \sum_{r\in\tau} \sum_{i\in N\cup\{0\}} \sum_{i'\in N\cup\{0\}} u_{ii'r} c_{ii'} + \sum_{i\in N} z_i o_i \tag{1}$$

$$\sum_{r=1}^{m} x_{jr} \leq 1 \qquad \forall\, j \in T \tag{2}$$

$$\left(\sum_{j\in T} x_{jr} \leq 1 \right) \qquad \forall\, r \in \tau \tag{3}$$

$$\sum_{r=1}^{\mu} y_{ir} + z_i = 1 \qquad \forall\, i \in N \tag{4}$$

$$\sum_{i\in N} u_{0ir} \leq 1 \qquad \forall\, r \in \tau \tag{5}$$

$$y_{ir} w^i_{\alpha\beta} \leq \sum_{j\in T} x_{jr}\, v^j_{\alpha\beta} \qquad \forall\, i \in N, \forall\, r \in \tau, \quad \forall\, \alpha \in (1,\dots,p), \forall\, \beta \in (1,\dots,q) \tag{6}$$

$$e_i \leq b_i \leq l_i \qquad \forall\, i \in N, \forall\, r \in \tau \tag{7}$$

$$\sum_{i\in N\cup\{0\}} u_{ii'r} = \sum_{i\in N\cup\{0\}} u_{i'ir} = y_{ir} \qquad \forall\, i' \in N, i \neq i', \forall\, r \in \tau \tag{8}$$

$$b_i + s_i + t_{ii'} \leq b_{i'} + M_1(1 - u_{ii'r}) \qquad \forall\, i, i' \in N, i \neq i', \forall\, r \in \tau \tag{9}$$

$$t_{0i} \leq b_i + M_2(1 - u_{0ir}) \qquad \forall\, i \in N, \forall\, r \in \tau \qquad (10)$$

$$b_i + s_i + t_{i0} \leq l_0 + M_1(1 - u_{i0r}) \qquad \forall\, i \in N, \forall\, r \in \tau \qquad (11)$$

$$x_{jr} \in \{0,1\} \qquad \forall\, j \in T, \forall\, r \in \tau \qquad (12)$$

$$y_{ir} \in \{0,1\} \qquad \forall\, i \in N, \forall\, r \in \tau \qquad (13)$$

$$z_i \in \{0,1\} \qquad \forall\, i \in N \qquad (14)$$

$$u_{ii'r} \in \{0,1\} \qquad \forall\, i, i' \in N \cup \{0\}, i \neq i', \forall\, r \in \tau \qquad (15)$$

$$b_i \geq 0 \qquad \forall\, i \in N \qquad (16)$$

$$M_1 = \max_{i \in N}(l_i) + \max_{i,i' \in N \cup \{0\}}(t_{ii'}) \qquad (17)$$

$$M_2 = \max_{i \in N}(t_{0i}) \qquad (18)$$

The objective function (1) minimizes the total travel costs and the outsourcing costs for the unscheduled tasks. Inequality (2) ensures for every available technician that he is not assigned to more than one team. Note that inequality (3) is only used for the case when one technician alone forms a team, thus the solution method without team building. Constraint (4) ensures that every task is either performed or outsourced, while inequality (5) makes sure that each team leaves the depot once only. Inequality (6) denotes that each task is assigned to a team which is able to meet its requirements. Inequality (7) sets the range for the beginning time between the earliest possible starting and the last possible ending time. Constraint (8) ensures that each task has a direct predecessor and a successor (also including the depot). Inequalities (9), (10) and (11) are responsible for the correct time sequence of the tasks, where inequality (9) ensures for each task that its direct successor cannot be started until it is finished and the team arrives at the next location. Inequality (10) bounds the earliest starting time for the first assigned tasks for every team. The teams leave the depot at time zero ($e_0 = 0$), therefore the service can begin at the earliest when they arrive ($0+t_{0i}$). Inequality (11) makes sure that all teams get home to their depot in time. The constraints (12) – (16) define the characteristics of the used variables. In (17) and (18) sufficiently big values are used for setting the variables M.

4. Related work

A very similar problem is dealt with in Xu and Chiu (2001) named *field technician scheduling problem* (FTSP). Different types of tasks with different requirements have to be executed by technicians with differing skills. The tasks are located at different locations and their visiting is constrained by time windows. In contrast to the current solution method the FTSP neglects the chance of forming teams. Furthermore, tasks can have different priorities since a repair job might be more important than a periodic maintenance. An aggravating factor is the working time window for each technician in which they are available.

To identify a good quality solution, three levels of objectives are used. The first objective is to schedule as many tasks as possible within the given time frames. Secondly, the working time, including travel and waiting time is minimized, whereas the third objective only pays attention to the total traveling time of the employees.

Additionally, the challenge organized by the French Operational Society (ROADEF) in 2007 handled the scheduling problems of the telecommunication company France Telecom (Dutot et al., 2006). The participants of the challenge faced a similar problem, where technicians with different skills have to perform tasks with varying difficulties and different priorities. Some tasks have higher priorities as they might be urgent repair tasks and others might have lower priorities because they are regular maintenance tasks. What´s more, some tasks depend on each other because of the existence of precedence constraints. Another differentiation is the budget for outsourcing which is not penalized in the objective function but may not be exceeded.

The problem however does not consider routing costs between the tasks. The solutions are graded by the objective function, which is the weighted sum of the ending times of the last tasks of each priority type. As the team of Cordeau et al. (2008) achieved a tied second place in the challenge by using a solution framework based on the *adaptive large neighborhood search* (ALNS), several ideas are implemented in the present solution method.

5. Solution method

Since the basic *vehicle routing problem* (VRP) is NP-hard it is clear that the *service technician routing and scheduling problem*, which is an extension of it, must be NP-hard as well. It is unreasonable to solve such problems exactly because this would take too long for being useable in real life situations. Therefore, the problem is tackled by using a heuristic approach which creates an initial solution and then improves it by applying a local search method.

An improvement phase is executed since simple construction heuristics are usually unable to find high quality results. Nevertheless, local search methods can improve the results as they try to find better solutions by iteratively exploring the neighborhood of the initial one. Hence, the solution quality depends largely on the initial solution itself and the way how neighborhoods are defined. Even though various neighboring solutions are compared, the descending characteristic of many local search approaches, which only accept improved solutions as new incumbents, might prevent moving towards the global best solution from a local optimum. This disadvantage is compensated by applying a metaheuristic framework, which also allows a deterioration of the solution to a certain extent.

The recently applied *adaptive large neighborhood search* achieved very impressive results when using it to solve VRP instances as demonstrated in Pisinger and Ropke (2007). What´s more, it was also able to perform very well when solving the *technician and task scheduling problem* (TTSP) (Cordeau et al., 2008) assigned during the ROADEF challenge. Since the current problem is a mix of the two mentioned problems, the ALNS was chosen to apply, in the hope that it also copes with the characteristics of this problem.

5.1 Adaptive large neighborhood search

The *adaptive large neighborhood search* proposed by Ropke and Pisinger (2006b), Pisinger and Ropke (2007) and Cordeau et al. (2008) is derived from the *large neighborhood search* (LNS) proposed by Shaw and the *ruin and recreate algorithm* (RR) described by Schrimpf et al. (2000). Both methods were applied to the VRP and

are based on the idea of continuously removing tasks from the routing plan (destruction) and reinserting them at positions where they ideally cause less costs (repair).
Given an initial solution obtained by a construction heuristic, the factors which affect the quality of these methods are the number of tasks to remove from the schedule and the decision how to destroy and repair the solution.

Traditionally, classic neighborhood search algorithms, with the corresponding neighborhood operators, are able to discover a lot of solutions in short time. Nevertheless, for large problem instances or instances which are highly constrained, they fail to achieve big improvements due to the constrained neighborhood size.
A relief for this weakness is the application of *very large-scale neighborhood search techniques* (VLNS) presented by Ahuja et al. (2002). Ahuja et al. analyzed the importance of the applied neighborhood structure and highlighted the positive relationship between the neighborhood size and the solution quality. However, a larger neighborhood is connected to longer execution times, which in turn results in fewer solutions found per time unit, unless very efficient search mechanisms are executed.
The neighborhood selection mechanisms of the LNS and RR algorithms could be interpreted as efficient in a sense that numerous possibilities for removing and reinserting visits are exploited to move freely in the solution space instead of exploring it in big segments.
So, in comparison to the classic neighborhood search algorithms, which try to obtain improvements by iteratively changing some attributes of the given solution, the described methods perform very powerful moves resulting in a large alteration of the solution.
The ALNS is extended in a way that various simple destroy and repair heuristics are used, while the LNS uses one destroy and one repair method. This extension enables the adaptation to different problem instances due to the fact that different sub-heuristics cause the ALNS to be able to operate on structurally different neighborhoods. Furthermore, the ALNS uses a *simulated annealing* (SA) procedure to avoid getting trapped in a local minimum.

5.2 Outline of the algorithm

Given an initial solution obtained by the construction heuristic, in the ALNS phase one destroy and one repair heuristic is chosen as described in Section 5.3.

First, the destroy heuristic is applied and the solution is destroyed by removing already scheduled tasks. The destroy sub-heuristics used are the *random destroy, worst destroy, related destroy* and *cluster destroy* as described in Pisinger and Ropke (2007) and the *team destroy* as used by Cordeau et al. (2008). Second, the unscheduled tasks are tried to be reinserted to repair the schedule and obtain a new solution. The applied repair sub-heuristics are the *greedy* heuristic and the *regret-q* heuristics as described in Pisinger and Ropke (2007). Additionally, the *insertion* heuristic suggested by Solomon (1987) was implemented.

The new solution obtained by the destruction and the subsequent repair is accepted to be the new current incumbent solution under certain circumstances. Even though a simple descent approach was used in the LNS which accepts improved solutions only, the ALNS is based on a *simulated annealing* framework developed by Kirkpatrick et al. (1983), which also allows a worsening of the solution with a certain probability.

The term simulated annealing originates from the discipline of statistical mechanics, which among others is concerned with the process of finding the ground state of a matter with low energy configurations. To obtain a clean crystal from a melt, the melted substance has to be cooled down slowly. This allows the atoms to find their equilibrium. A fast cool down, also called as quenching, would cause the substance to get out of equilibrium, resulting in a crystal which would have many defects. The link to the combinatorial optimization arises by comparing the energy state with the value of the objective function. Accepting only improved solutions during a local search is like a rapid quenching from high temperatures, so it is obvious that the obtained results are usually metastable. The *simulated annealing* is applied to the underlying solution method in a way that as a result better solutions are always accepted to be the current incumbent. But to escape from a local optimum, even an increase of the objective value is accepted with a probability calculated as:

$$e^{-(f(x')-f(x))/T}$$

The acceptance criterion is controlled by the temperature T and the cooling rate c, whereby f(x´) denotes the objective value of the new solution x´ and f(x) the objective value of the current incumbent solution x. The solution is "melted" at a high temperature and in every iteration the temperature is lowered linearly with $0 < c < 1$ so in the following iteration $T = Tc$. To define the melting point T_{Start}, we refer to Ropke and Pisinger (2006b), who calculated it in dependency of the initial solution instead of setting a fixed value disregarding the problem instance. Nevertheless, the initial solution is modified in a way that the outsourcing costs are neglected. The reason for doing so is that travel costs are small in comparison to the outsourcing costs, so if the construction heuristic would not be able to schedule all tasks, T_{Start} would be set to very high values. Let $f(x)_{modified}$ denote the modified initial objective value, then T_{Start} is calculated in such a way that a solution which is w_T percent worse than the initial solution is accepted with probability 0.5.

$$T_{Start} = -\frac{w_T}{\ln 0.5} f(x)_{modified}$$

The selection of the destroy and repair heuristic is performed stochastically by applying a *roulette wheel selection* mechanism. Nevertheless, every pair of destroy "a" and repair "b" heuristics is weighted in the weight matrix ρ_{ab} according to its combined performance. The better a pair of methods performs, the higher its weight and therefore the probability of getting chosen in the following iterations. This is due to the fact that the probabilities are calculated on the base of the weights. The heuristic selection is explained in the next section in more detail.

Algorithm 1 outlines the pseudo code of the ALNS process in which variable f(x*) represents the objective value of the global best solution x*.

Algorithm: 1

```
1   input: feasible solution x created by the construction heuristic
2   x* = x; ρab = (1, …, 1)
3   repeat
4       choose a pair of destroy and repair heuristic (a and b) by using the roulette
        wheel selection (Section 5.3)
5       x´ = b(a(x))  // apply the heuristics to solution x to obtain x´
```

6 **if** f(x´) < f(x*)

7 x = x* = x´

8 **else if** accept(x´, x)

9 x = x´

10 record the solution quality and update ρ_{ab} as described in Section 5.3

11 **until** stopping criterion is met (number of iterations)

12 **return** x*

The check of the time window feasibility of any solution is performed by using Solomon´s (1987) lemma. It is applied every time the algorithm tries to insert a task into the schedule and can be described as follows.

Before inserting task h into a team´s route, first, the compliance of its time window constraint is verified. Moreover, h´s effect on all subsequent tasks in the route must be considered. For all subsequent tasks, a push forward value PF_i is calculated, which declares by how much the starting times are delayed because of the insertion of h. Task h is only scheduled to a team if the starting times of all its successors are within the respective time windows and the team can return to the depot before it is closed.

The effect on h´s successors is calculated as

$$PF_i = b_i^{new} - b_i \qquad \text{for task i = h´s direct successor} \cup \{0\}$$

whereas b_i^{new} denotes the new starting time of i under the assumption that h is scheduled before it.

$$PF_{i'} = \max\{0, PF_{i'-1} - idel\ time_{i'}\} \qquad \forall \text{ i´ scheduled after i} \cup \{0\}$$

The idle time at task i´ is the time that appears if a team arrives to task i´ before its time window has started.

The insertion of h is feasible if:

$$b_h \leq l_h$$

and

$$b_i + PF_i \leq l_i \qquad \forall\ i \in u's\ successors$$

5.3 Selection of the repair and destroy heuristics

Given that several sub-heuristics are available, one destroy and one repair method must be chosen to obtain a new solution. This selection happens randomly, but the probability for choosing a heuristic depends on its performance in the former applications. The applied selection approach is similar to that used by Ropke and Pisinger (2006b) and Pisinger and Ropke (2007) but it contains a few modifications. While the mentioned algorithms select the destroy and repair heuristics independently from each other, this approach links them together to identify good pairs of heuristics. The probability of choosing a heuristic pair is controlled by the matrix ρ_{ab} that indicates the weight for choosing destroy method "a" together with repair method "b". If the number of available destroy heuristics is denoted with η^d and the number of repair heuristics with η^r, then the probability ϕ_{ab} for choosing the pair "a" and "b" is calculated as:

$$\phi_{ab} = \frac{\rho_{ab}}{\sum_{k=1}^{\eta^d} \sum_{l=1}^{\eta^r} \rho_{kl}}$$

The actual heuristics are chosen by a *roulette wheel selection* mechanism. It can be imagined as a wheel where all possible heuristic pairs (η^d x η^r) are represented by a slice of it, with the size being determined by the corresponding probability. The higher the probability the bigger is the slice. The wheel is spun just like a roulette wheel and the heuristic pair on which the "ball" comes to rest is applied.

In contrast to the LNS, in which only one removal and one insertion approach is applied, this type of selection enables the algorithm to adjust itself to various problem types. It does this by simply weighting more appropriate heuristics higher than those heuristics which are not able to push the solution search further.

To adjust the weights automatically, the *adaptive weight adjustment algorithm* proposed by Ropke and Pisinger (2006b) is used.

The weights on the base of which probabilities are determined are equal in the beginning and are set to one. Nevertheless, during the search, pairs of heuristics can earn higher weights by providing better solutions. After every iteration of the ALNS heuristic, the solution is analyzed and the scores ψ_{ab} for every pair of sub-heuristics are updated by increasing them by either σ_1, σ_2 or σ_3. Each parameter corresponds to a certain category of solution quality.

In the first category, the obtained solution is the new global best solution, so the score is increased by parameter σ1. In the second category, the score is increased by σ_2 due to the fact that the application of the destroy and repair heuristic led to a solution that is better than the current incumbent and it has not been visited before. Finally, the third category which contributes σ_3 to the score, is characterized by a solution that is worse than the current incumbent, but it was accepted because of corresponding to the simulated annealing criterion and it has not been visited before.

$$\psi_{ab} = \begin{cases} \psi_{ab+}\,\sigma_1 & \text{if the last destroy-repair operation obtained a new global best solution} \\ \psi_{ab+}\,\sigma_2 & \text{if the last destroy-repair operation yielded a solution that is better than the current incumbent and it has not been accepted yet} \\ \psi_{ab+}\,\sigma_3 & \text{if the last destroy-repair operation yielded a solution that is worse than the current incumbent and has not been accepted yet but is accepted now} \end{cases}$$

Of course, heuristics that generate improved solutions are preferred. Nevertheless, it makes sense to remunerate all heuristics that are able to identify unvisited solutions and diversify the search, because such heuristics help to explore the solution space at a higher level.

This is true for the σ_3 cases: even though the heuristics cannot create a good solution, they bring the search forward so they earn an increase in the score.

The evaluation of the sub-heuristics is divided in segments of 100 iterations of the ALNS heuristic. The scores ψ_{ab} and the counter Θ_{ab}, which counts how many times a pair of heuristics has been applied, are all initialized with zero at the beginning of each segment and are updated after every destroy-repair operation until the segment is finished. Then, the new weights ρ_{ab} and the new probabilities ϕ_{ab} derived from them are calculated for the next segment by using the captured data.

At the beginning of the first segment, all weights are initialized with one. Therefore, during the first 100 iterations the heuristics are chosen with the same probability. Afterwards, when a segment is finished the new weights are calculated as follows:

$$\rho_{ab} = \rho_{ab}(1 - \mathrm{r}) + \frac{\psi_{\mathrm{ab}}}{\Theta_{\mathrm{ab}}}\mathrm{r}$$

Parameter r ∈ [1; 0] is the *reaction factor* that controls how sensitive the weights are to changes in the heuristic pair performance and derived from that, to changes in the scores. The extreme case of setting r equal to zero would ignore the performance of the heuristics and the weights would remain unchanged, whereas r equal one means that the historic performance has no impact. A reasonable setting for r should be selected in-between the two extremes.

Note, that only the weights for those destroy-repair method pairs are changed that have been used during the respective segment.

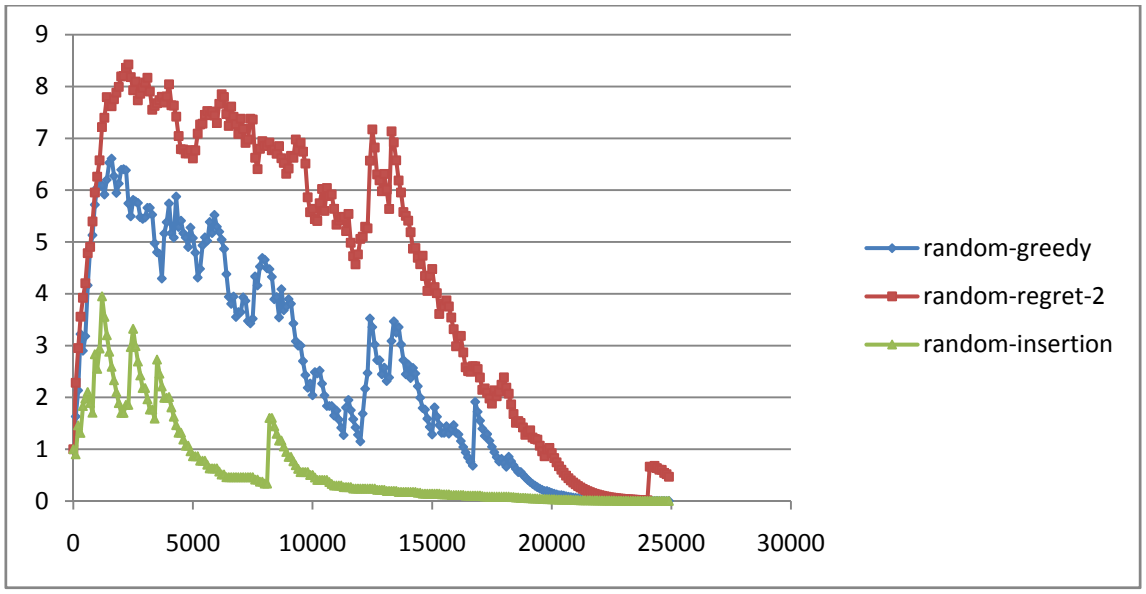

Figure 1

Figure 1 illustrates an example of the weight development (y-axis) of three different pairs of heuristics during 25000 iterations (x-axis). The compared pairs are random-greedy, random-regret-2 and random-insertion heuristics. The problem instance used for this run is R101_5x4_noTeam with the complete set of technicians (see Section 9).

All weights are set to one initially, but during the run it becomes apparent that the random-regret-2 curve earns the highest weight and dominates the compared method pairs almost over all iterations. The pair random-greedy performed slightly worse than the random-regret-2 pair, while the heuristic combination random-insertion performed worst. Based on this graph, it can be said that the random-insertion heuristics are used much rarely than the other two pairs.

The shape of the graphs can be explained by the fact that in the beginning it is easier to improve the solution and to identify unvisited solutions. Therefore, the heuristics are

rewarded with higher scores ψ_{ab}, which implies a higher positive slope of the curves. Later, when it becomes much harder to improve the solution or to find unvisited ones, the heuristics are rewarded lower or not at all. This results in a downward slope of the weight development curves.

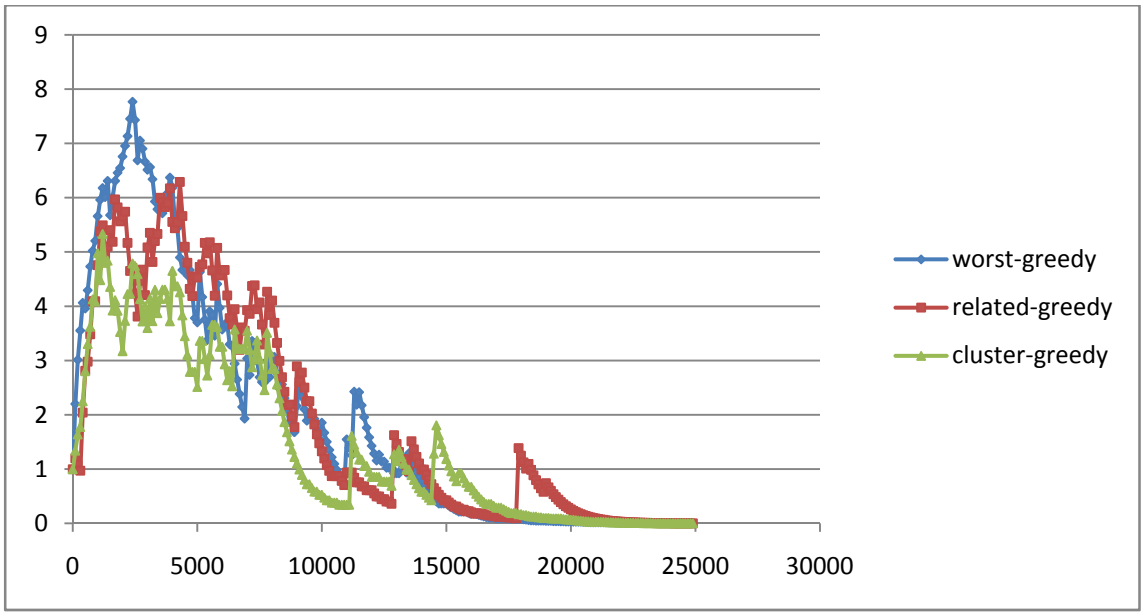

Figure 2

The data for Figure 2 were collected during the same run of the ALNS as for Figure 1, but in this graph the weight development of heuristic pairs, which use the same repair heuristic, namely the greedy approach, are compared. In the beginning, all pairs perform quite equally, but from iteration 1200 the cluster-greedy pair cannot achieve as good results as the worst-greedy or the related-greedy methods. At the end of the search the performances are evened again. Therefore, all the compared pairs have almost equal probabilities of getting chosen in the roulette wheel selection except during iterations 1200 to 6000. In this state the cluster-greedy pair is used more scarcely.

Given the different destroy-repair heuristic pairs, in principal, a local search with several neighborhood structures is applied in the ALNS. Therefore, some conformities can be identified when comparing the ALNS to the *variable neighborhood search* (VNS) proposed by Mladenović and Hansen (1997). Based on an initial solution, which is set to be the current incumbent, the VNS applies different neighborhood operators.
In every iteration, one of the available neighborhood types is selected and used to obtain a set of new solutions. One of them is chosen randomly and is optimized in a further

local search phase. Whether the achieved solution is accepted as current incumbent or not, depends on the user's solution evaluation approach.

6. Version without team building

In this thesis two algorithm versions are distinguished: the first, in which tasks require only one technician, so team building is not allowed (or one team consists of only one technician) and the second, in which tasks may require more technicians, so technicians must be grouped together to combine their skills.
In this chapter we focus on scheduling problems, in which no task requires more than one technician. Therefore, technicians move and work alone. Even though technicians are not grouped, every technician represents one team. Or in other words, teams execute tasks, whereas one team consists of exactly one technician.

6.1 Construction heuristic

6.1.1 Introduction

This section describes how an initial solution is generated with an easy heuristic, which is also used later in the improvement phase with further heuristics. The aim is to select tasks and to schedule them repeatedly until a feasible and preferably low cost solution is created.
Basically, two types of construction heuristics are distinguished for the VRPTW, namely sequential and parallel tour building heuristics. Sequential methods build one route at a time, whereas parallel methods work on several - or in our case - on all available routes simultaneously. Given that the minimization of the number of routes is not relevant, each available team represents one route. In the first phase, empty routes are generated for every team and in the second phase, these empty routes are filled up with the unscheduled tasks by applying a greedy approach. Even though, Pisinger and Ropke (2007) applied the *regret-2* heuristic to create the initial solution, in this solution method the greedy approach is used. This is done because to be able to calculate a regret

value at least two routes are necessary. Nevertheless, in some instances one route might be sufficient to cover all tasks.
This type of solution method is referred to as the version without team building in the following, as technicians are not grouped together.

6.1.2 Greedy heuristic

The *greedy* heuristic is a parallel route building method and was used by Ropke and Pisinger (2006b) and Pisinger and Ropke (2007) as a repair heuristic in the improvement phase. It works simultaneously on the number of routes m, which is given by the number of teams. As mentioned above one team consists of one technician in this version. The method checks every feasible insertion position for every task and inserts the task at the position which produces the least increase in the objective value. This process continues until either all tasks have been scheduled or no more tasks can be inserted without violating some constraints.
Formally spoken, all $\Delta f_{i,r}$ values, which denote the change in the objective value if task i is assigned to team r at its cheapest position, are kept. Subsequently, the task which produces the least increase or the biggest decrease in the objective value is scheduled at its minimum cost position.

$$(i, r) = arg \min_{i \in N, r \in (1,..,m)} \Delta f_{i,r}$$

In order to save computing time, the characteristic of changing only one route per iteration is exploited and only those $\Delta f_{i,r}$ values are recalculated which are affected by the previous insertion.

6.2 Solution improvement

To improve the initial solution, the ALNS is applied. The algorithm iteratively destroys the solution by removing some of the scheduled tasks to reinsert them at better positions. To do so, five destroy heuristics and three basic repair heuristics are available. The method pairs to apply are chosen randomly on the base of their

corresponding weights, earned during former iterations. Whether the new solution, achieved by the applied heuristic pair, is accepted to be the next current incumbent or not is decided according to the SA criterion. In the following, the available destroy and repair heuristics are described.

6.2.1 Destroy heuristics

In every iteration, one of the four destroy methods proposed by Ropke and Pisinger (2006b) and Pisinger and Ropke (2007) is applied to remove a set of already scheduled tasks. Every method has a different approach and tries to react to the various aspects of the problem sets and therefore makes it easier to find better results.

When tasks have been removed, it is important to compress the solution in a way that the already scheduled tasks are tried to be moved to an earlier position for every team, taking into account the time window constraints. This compressing process was suggested by Cordeau et al. (2008) to avoid too big holes in the schedule. This is possible, as by executing the repair methods, tasks can be inserted in every feasible position anyhow by pushing the successor tasks forward.

Before starting the destruction, u, the number of tasks to remove must be set. This number is chosen randomly from the range of min $\{0.1n_a, 30\}$ and max $\{0.4n_a, 60\}$ whereas n_a denotes the number of assigned tasks, as suggested by Pisinger and Ropke (2007). This means that for large instances the number of tasks to remove is between [30, 60] and for small instances between $[0.1n_a, 0.6n_a]$. Removing more than 60 tasks in each iteration would lead to solutions which are rarely accepted, because of the weak repair methods. On the other hand, removing less than $0.1n_a$ would not contribute much to an improvement of the solution either.

6.2.1.1 Random destroy

The *random destroy* is the simplest method, but it helps to diversify the search. This method continues to remove scheduled tasks randomly until the earlier set number of tasks to remove u is met.

6.2.1.2 Worst destroy

Worst destroy is an approach which pays attention to the objective function and so removes tasks which deteriorate it the most because of a wrong positioning. To find out which task contributes the most to a bad solution, for every task i its cost of insertion (cost(i,x)) is calculated. This insertion cost is defined as the difference between the solution with task i (f(x)) and the solution when task i is removed (f_{-i} (x)):

$$cost(i,x) = f(x) - f_{-i}(x)$$

It is obvious that tasks with high cost(i,x) should be removed to be rescheduled at a cheaper position. The heuristic repeatedly chooses a task and removes it until u tasks have been removed as demonstrated in Algorithm 2. Nevertheless, it is not always the task with the highest cost(i,x) that is removed but a randomized selection with a degree of randomization controlled by the parameter p_{worst} is used. All scheduled tasks are listed in the array L sorted in descending order of cost(i,x). A random number y in the interval [0,1] is drawn and the task in the $L[y^{p_{worst}}|L|]$ position is chosen to be removed. The larger the parameter p_{worst}, the higher the probability that the task with the highest cost(i,x) is selected.

Without this type of selection, the same set of tasks would be removed again and again, provided that the input solution for the *worst destroy* method is the same. Therefore, no improvement in the solution could be expected.

Algorithm: 2

```
1   input: feasible solution x
2   removed = 0
3   repeat
4           array L = scheduled tasks i sorted in decreasing order of cost (i,x)
5           choose random floating number y in the interval (0,1)
6           select task:  i = L[y^p_worst |L|]
7           remove task i
8           removed++
9   until removed >= u
10  compress solution
```

6.2.1.3 Related destroy

The *related destroy* approach was introduced by Shaw (1997, 1998). The main idea is to remove related tasks which will help to create more opportunities for interchange in the reinsertion phase or in other words, remove tasks that can easily be exchanged. By removing dissimilar tasks, they probably would be reinserted at their former position, so no improvement could be expected.

Relatedness is defined as geographical closeness, closeness in the starting time of the service but also as the similarity of the task requirements. Therefore, two tasks are related, if they have a small distance to each other, their current service starting times are scheduled close and they require similar technician skills. The skill similarity $\gamma_{ii'}$ between two tasks i and i´ is used as proposed by Cordeau et al. (2008) and is obtained by comparing their skill requirement matrices $w^{i}_{\alpha\beta}$

$$\gamma_{ii'} = \sum_{k=1}^{p}\sum_{l=1}^{q}\left|w_{kl}^{i} - w_{kl}^{i'}\right|$$

The three criteria are combined in the following formula to obtain the weighted relatedness measure R(i,i´) between task i and i´. The weights are indicated by the respective w parameters.

$$R(i,i') = w_{distance}d_{i,j} + w_{time}\left|b_i - b_j\right| + w_{similarity}\gamma_{ij}$$

So, the lower R(i,i´), the more related the two interventions are.

The heuristic first selects and removes a task i randomly from the schedule. Then the set of removed tasks S is initialized with i. On the base of the chosen task, the relatedness to all other scheduled tasks is calculated and one with a lower R(i,i´) measure is chosen to be removed. Again, the algorithm is diversified by adding some randomness into the choice. Tasks are sorted in list L in increasing order of relatedness and the task in the $L[y^{p_{related}}|L|]$ position is removed from the solution and inserted into the set S. Parameter $p_{related}$ again controls the degree of the random influence to the selection and y is a random number in the interval [0,1]. The next task on the base of which the R(i,i´) values are calculated is chosen randomly from S and the algorithm is continued until u tasks have been removed as in Algorithm 3.

Algorithm: 3

```
1   input: feasible solution x
2   choose a scheduled task i randomly
3   remove task i and initialize set S of removed tasks S = {i}
4   compress solution //by removing a task the starting time of the remaining changes
5   removed = 1
6   repeat
7       i* = task randomly selected from S
8       array L = scheduled tasks i not in S sorted by increasing R(i,i*) values
9       choose random floating number y in the interval (0,1)
10      select task:  i = L[y^p_related |L|]
11      S = S ∪ {i}
12      remove task i
13      compress solution
14      removed++
15  until removed >= u
```

6.2.1.4 Cluster destroy

Cluster destroy (Ropke and Pisinger, 2006a) is based on the *related destroy* heuristic where some kinds of related tasks are selected to be removed. However, it is not the first priority in this heuristic to make it easier to interchange them among each other, but to spot and remove clusters of tasks which are on the same route instead of removing fewer tasks from more routes. In this method, the term related refers to spatial closeness, so local coherent tasks are removed. The reason for not using the *related destroy* but implementing a variant of it, is that it is important to remove the whole cluster with all its associated tasks on a given route at once, otherwise there would be a tendency to reinsert the removed tasks back to their former position. The *cluster destroy* selects a geographical cluster for a certain route and removes it entirely. Then, the removed tasks can either be distributed to different routes or the whole cluster can be integrated into a close route.

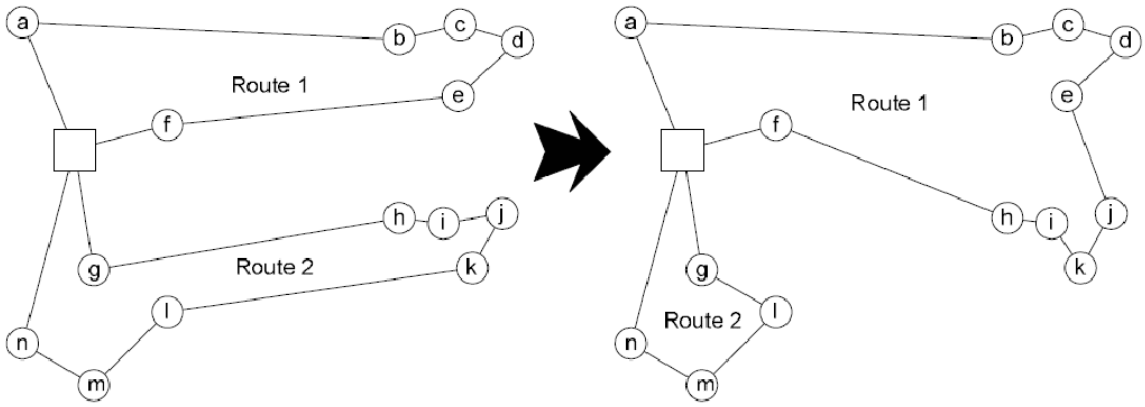

Figure 3

Figure 3, assumed from Ropke and Pisinger (2006a), depicts an example for the cluster removal with a subsequent repair. The square stands for the depot whereas the circles represent the individual tasks. In the upper example the cluster removal heuristic is applied to route 2. As a next step, the route is divided into two clusters. The first, consisting of task g, l, m and n and the second, consisting of tasks h, i, j and k. The heuristic chooses one of the clusters with a probability of 0.5 and removes all tasks in a cluster, in the upper example tasks h - k. The removal of the entire chunk of tasks makes it possible for the repair heuristic applied afterwards to create a schedule with lower routing costs. Even if one of the tasks in the cluster would be retained in route 2, as could happen in the related destroy, the repair heuristics would tend to reinsert the removed tasks back to their former position.

For identifying clusters, Kruskals algorithm (1956) for identifying the shortest spanning subtree is applied. By deleting the longest edge of the minimum spanning tree, two clusters are obtained. The clustering is embedded into the removal heuristic as follows. The first team whose route is destroyed is selected randomly. After indentifying two clusters of tasks, one of them is chosen with a probability of 0.5 and all tasks in this cluster are removed. If the number of tasks to remove is achieved, the algorithm stops. Otherwise, one of the current removed tasks is chosen randomly (i*) and the closest scheduled task on a different route is selected. The route of the new task is than divided in two clusters and one of them is removed. This process continues until the number of removed tasks reaches u or no more clusters can be found because of too short routes. Algorithm 4 shows the pseudo code of the described procedure.

Algortihm: 4

```
1   input: feasible solution x
2   choose team j randomly which has at least 3 assigned tasks
3   removed = 0
4   repeat
5       find the minimum spanning tree between the assigned tasks of team j
        (Kruskal`s algorithm)
6       remove the largest edge to obtain two clusters
7       select one of the clusters with probability 0.5 and remove all tasks
8       removed += |removed cluster|
9       select task i* of selected cluster randomly
10      find scheduled task i which is spatially closest to i*, is in a team different
        from j and has at least 3 assigned tasks
11      j = team of i
12  until removed >= u or there are no more teams with at least 3 assigned tasks
13  compress solution
```

6.2.2 Repair heuristics

After the solution is ruined by one of the available destroy heuristics, the removed tasks together with the residual unscheduled tasks are tried to be reinserted by one of the repair heuristics described below. In the improvement phase, parallel and sequential methods are used side by side, since both of them have specific advantages when applying them on different problem sets. Potvin and Rousseau (1993) came to the conclusion that the parallel approach performs better than the sequential one when solving VRPTWs with exclusively random, and a mix of clustered and random distributed customers. In turn, the sequential method outperforms the parallel for exclusively clustered problems. To combine the strengths of both methods, sequential and parallel methods are used together in the improvement phase.

6.2.2.1Greedy heuristic

Just like described above in the solution construction phase, the *greedy* heuristic is used to repair the solution by repeatedly inserting tasks in their cheapest possible route at their best position.

6.2.2.2 Insertion heuristic

The *insertion* heuristic used in this solution framework is a slight modification of Solomon´s (1987) insertion method called I1. It differs from the greedy and regret approach in a way that it is a sequential route construction heuristic. It considers only one route at the same time and only continues with the next if there is no possibility to insert another task. Furthermore, the decision which task to include is based on multiple criteria in addition to the objective function.

The heuristic for the VRPTW first initializes a route with a seed customer and then adds the remaining unscheduled customers, by considering the respective constraints, into the route until it is full. If there are still unscheduled customers, a new route is initialized.
This procedure is continued until all customers are serviced. The seed customers to initialize a new route are either the geographically farthest unrouted customer or the unrouted customer with the earliest deadline.
However, the initialization process in this heuristic framework is omitted because of two reasons. First, the heuristic is used as a repair heuristic, so it faces existing routes which in most of the cases are initialized anyway. Second, in cases, in which the destroy heuristics remove all tasks from a route, the insertion criteria on the base of which tasks are assigned are good enough to initialize the routes. Another difference to the original solution method is the restriction on the number of routes derived from the number of teams.

This method uses two criteria $c_1(i, h, i')$ and $c_2(i, h, i')$ to decide which customer h is inserted at which position between two assigned and adjacent customers i and i´. In the case of empty routes, i and i´ refer to the depot, so inserting h between them would result in a pendulum route. The cost for inserting h between i and i´ is reflected by $c_1(i, h, i')$, which is defined as the weighted average of the extra travel cost $c_{11}(i ,h, i')$ and the extra time needed for the insertion $c_{12}(i ,h, i')$. In the current heuristic, factor $c_{11}(i ,h,$

i′) is modified to be the extra cost of the objective function when inserting h, to account for the outsourcing costs as well. Factor $c_{12}(i, h, i')$ is defined as the difference between the new starting time for service at task i′ ($b_{i'}^{new}$) when h is inserted between i and i′ and the staring time before inserting h ($b_{i'}$).

The criteria $c_1(i, h, i')$ and $c_2(i, h, i')$ are calculated as follows

$$c_1(i, h, i') = \alpha_1 c_{11}(i, h, i') + \alpha_2 c_{12}(i, h, i')$$

where

$$\alpha_1 + \alpha_2 = 1 \quad \alpha_1 \geq 0, \ \alpha_2 \geq 0$$

$$c_{11}(i, h, j) = d_{ih} + d_{hi'} - d_{ii'} - o_h$$

$$c_{12}(i, h, i') = b_{i'new} - b_{i'}$$

$$c_2(i, h, i') = \lambda d_{0h} - c_1(i, h, i')$$

Factor $c_2(i, h, i')$ depends on the insertion cost for inserting h between i and i′ and the distance from h to the depot. It is easy to observe that we want to insert customers with high $c_2(i, h, i')$ values, as they stand for the benefit derived from servicing a customer on the current route rather than on a direct route. Parameter λ defines how much the best insertion position of h depends on its distance from the depot. Furthermore, λ adjusts the weighting between the mentioned distance and the insertion cost.

6.2.2.3 Regret heuristic

The *regret* heuristic was proposed by Potvin and Rousseau (1993) for the VRPTW by adopting the insertion framework of Solomon (1987). Furthermore, Ropke and Pisinger (2006b) and Pisinger and Ropke (2007) used this heuristic in the ALNS framework.
It is a parallel approach which is based on a regret measure calculated for all unrouted customers.
The *regret* heuristic develops the greedy approach further in order to overcome its weaknesses in identifying difficult tasks which have only a few feasible and reasonable insertion possibilities. As the greedy heuristic inserts tasks at their current best cost position, it ignores that by doing so, the placement of more challenging tasks is delayed

to later iterations where the possible insertion positions are either limited or nonexistent. The *regret* heuristic reacts to this problem by adding a kind of forecast measure to the insertion decision. This measure named *regret* is calculated for every task and is an indicator of by how much the solution quality would suffer later if we wouldn´t insert a task at its best position now. As the Potvin and Rousseau approach is based on Solomon´s insertion heuristic, they define the value of the regret as the "gap between the best insertion place for a customer and its best insertion place in the other routes", whereby the best insertion places are determined by the insertion cost formula developed by Solomon. In our context, *regret* is defined as in Ropke and Pisinger (2006b).

If the variables Δf_i^q represent the change in the objective value when task i is inserted into its q^{th} cheapest route at its best position, then in each iteration, the regret value for task i is defined as:

$$regret_i = \Delta f_i^2 - \Delta f_i^1$$

Regret values therefore indicate the difference in the cost of inserting task i at the best position in its cheapest route, and the best position in its second cheapest route. The infeasibility of scheduling a task in a route would result in $\Delta f_i^q = \infty$, which in turn would cause tasks which have the fewest feasible routes for a potential insertion to have the highest *regret*.

Because we want to insert difficult tasks earlier to consider that they have fewer reasonable alternatives for insertion, the task with the biggest *regret* is inserted in its best route at its best position.

$$i = arg \max_{i \in N} (\Delta f_i^2 - \Delta f_i^1)$$

In the case of facing more tasks with equal *regrets*, the task which produces the lowest insertion cost is assigned just like it would happen in the greedy approach. The process continues until either all tasks are scheduled or no feasible position for unscheduled tasks can be found.

Pisinger and Ropke (2006b) refer to the extension of the basic *regret-2* heuristic, in which the best and the second best routes are crucial to a set of *regret-q* heuristics. The *regret* in *regret-q* heuristics is calculated as:

$$regret_i = \sum_{h=2}^{q} \Delta f_i^h - \Delta f_i^1$$

Again, the task with the highest regret value is inserted at its lowest cost position according to:

$$i = arg \max_{i \in N} \left(\sum_{h=2}^{q} \Delta f_i^h - \Delta f_i^1 \right)$$

Regret-q heuristics with q > 2 involve more than the best two routes in the decision. This enables an earlier identification of tasks which have only few good alternatives for insertion, so those are scheduled with higher priority. In other words, the higher q the further the heuristic looks ahead. Therefore, it detects earlier big insertion cost differences. On the other hand a *regret-1* heuristic focuses only on one route, therefore the regret is ignored and tasks are selected in decreasing order of their best insertion cost. Hence the solution obtained by the *regret-1* heuristic equals the solution achieved by the greedy approach. When applying different regret heuristics, attention should be paid to the number of existent routes. In either case the number of routes must be bigger or equal q. For the current algorithm *regret-2*, *-3*, *-4* and *-m* heuristics are used in dependency of the number of teams.

7. Version with team building

It happens frequently in real life situations that one encounters tasks that are too complex or too labor-intensive to be serviced by a single technician even if he is fully trained.

To handle such situations, several technicians must be grouped together so that their aggregated skills are sufficient to execute the assigned tasks together. As defined in Dutot et al. (2006), teams must stay together for a whole day, because the increased complexity and time consumption of mixing teams during a shift would make it ineffective to separate them. This is true since all technicians whose teams are affected must meet each other at the same time and the same location to enable a reassignment.

The condition that teams can´t be separated calls for a sophisticated solution method that is able to construct adequately arranged teams that ideally have low waste of skills.

Let´s assume that a team is built in order to serve a high number of tasks, but one of the technicians assists only during the execution of one intervention. The mentioned technician's skills would be wasted, except for this one task. So the quest is to innately form reasonable teams to avoid that some technicians have to kick their heels almost all day long, while their resources would be needed on other fields.

This chapter outlines the extensions made on the solution method described above to cope with variable team configurations in the construction, as well as in the ALNS phase.

7.1 Construction heuristic

As tasks require more than one technician, the first aim is to form teams in a way that their aggregated skill configuration enables the assignment of various interventions. Once the team building phase is completed, the greedy heuristic is applied to assign the unscheduled tasks as a second step.

A sophisticated approach for grouping technicians is the team construction method proposed by Cordeau et al. (2008), which is based on the selection of some seed tasks on the base of which teams are arranged. The algorithm is slightly modified to comply with the current problem's characteristics, but the basic idea to choose a task and then assign technicians one by one to an empty team until the resulting team is able to cover all of the tasks requirements is unchanged.

As we want the resulting teams to be heterogenic to be able to work on various tasks and to be sufficiently skilled to serve more complex tasks as well, seed tasks are determined by considering two criteria: difficulty and similarity in terms of the tasks´ requirement matrix $w^i_{\alpha\beta}$.

Difficulty β_i is a measure of how hard it is to create a team for a task. It is derived from the requirement matrix $w^i_{\alpha\beta}$ and is defined as:

$$\beta_i = \sum_{k=1}^{p} \sum_{l=1}^{q} w_{kl}^i$$

This measure provides information about the number of technicians needed, but also about the demand of technicians' skills. Consider two tasks A and B with the corresponding requirement matrices below. Even though both tasks require only one

technician, task B is more difficult (β_B=3) than task A (β_A=1), because a higher skilled technician is required to execute task B.

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

As heterogeneous teams with different skill configurations are able to serve a higher variety of tasks, the heuristic tends to choose seed tasks which differ from each other a lot. The actual value, by how much two tasks differ from each other, is defined by comparing their skill requirement matrix $w^i_{\alpha\beta}$ just like in Section 6.2.1.3. Nevertheless, it is not enough to define the similarity between only two seed tasks, since depending on the number of technicians, more teams may be constructed, which requires more seed tasks. Let S represent the set of already chosen seed tasks, then the similarity of task i to the whole set is defined as:

$$\gamma_{iS} = \sum_{i' \in S} \gamma_{ii'}$$

The two measures are combined to form a score for each task i defined as:

$$f(i,S) = w_\beta \beta_i + w_\gamma \gamma_{iS}$$

Variables w_β and w_γ are parameters which control the impact of each measure on the selection of the seed tasks.

The f(i,S) values are calculated for all unscheduled tasks. Nevertheless, because S is empty at the beginning, in the first iteration the scores are determined by only considering the difficulty. However, as seed tasks are added to S the f(i,S) values have to be recalculated as well. Tasks are sorted in list L in decreasing order of their corresponding score and the one with the highest f(i,S) is selected to build a team for it. If there is no possibility to form a team, the heuristic continues with the next task on the sorted list until a team can be created which is able to perform the selected task. The task is declared as a seed task, it is included in set S and scheduled to the new team. The f(i,S) values are recalculated by excluding the already chosen seeds and those tasks for which no team can be arranged because of the lack of technicians. Then the task with the highest score is selected again for the team creation. This process continues until either all tasks have been scheduled or the number of assigned technicians is greater or

equal $\chi|T|$. $\chi \in [0,1]$ can be defined as a safety stock of technicians that can be used in the following scheduling phase to reinforce the existent teams.

At first sight it seems that the outsourcing costs which have a big impact on the objective function are neglected. Nevertheless, we assume that difficult tasks also have higher outsourcing costs, so they are considered in the difficulty criterion implicitly. On the other hand, even if easy tasks would have high outsourcing costs because of some reasons, it would be counterproductive to consider these costs explicitly. Creating teams on the base of easy tasks, even if they have high outsourcing costs, would cause the teams to be too understaffed to serve a high variety of interventions.

7.1.1 Initial team composition

The process of forming teams follows a greedy approach (Cordeau et al., 2008) where technicians are assigned to teams that are dedicated to serve the potential seed tasks. The assignment happens according to the compliance of the skills demanded and offered. The initial point is an empty team which is gradually filled up with technicians until the resulting team is able to cover all of the required skills. The choice of including technicians into the team is based on their ability to cover preferably many of the required skills which are not covered yet by already assigned technicians. For exemplifying the team building procedure, the example used by Cordeau et al. (2008) is demonstrated.

Matrix A represents the requirement matrix of a task which is chosen to have a team built for it.

$$A = \begin{pmatrix} 1 & 2 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \end{pmatrix}$$

Three technicians are available with the respective skill matrices B, C, D:

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The following formula is used to calculate, how many skills a technician j with skill matrix $v^j_{\alpha\beta}$ covers of a task i´s skill requirement matrix $w^i_{\alpha\beta}$:

$$\sum_{k=1}^{p}\sum_{l=1}^{q} min\{v_{kl}^{j}, w_{kl}^{i}\}$$

For the upper example, the resulting covering scores would be 5, 7 and 2, respectively.

Technician C is able to perform the most out of task A's demands, so he is the first to be included in the team. Only those requirements which have not been handled yet must be considered in the following steps, so matrix A is updated to the actual requirements. Formally spoken, the new matrix A´= ($w'^{i}_{\alpha\beta}$) is calculated by setting

$$w'^{i}_{\alpha\beta} = \max\left\{0, w_{\alpha\beta}^{i} - v_{\alpha\beta}^{j}\right\}$$

The updated requirement matrix A´ is

$$A' = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The new covering scores are 4 by using matrix B and 1 by using matrix D. Accordingly, technician B would be assigned to the team. The combined skills of technician B and C are sufficient to execute task A.

In the case of more technicians having the same covering scores, a second criterion is introduced. The measure *waste* is defined as the unused or wasted skills that occur at assigning a technician to a task and a team.

$$\sum_{\alpha=1}^{p}\sum_{\beta=1}^{q} max\{0, v_{\alpha\beta}^{j} - w'^{i}_{\alpha\beta}\}$$

The resulting waste scores for the technicians with skill matrices B, C and D when assigning them to perform the task with requirement matrix A would be 2, 2 and 0 respectively. The team creation is illustrated in Algorithm 5.

Algorithm: 5

```
1  input: set of available technicians, set of unscheduled tasks
2  set of seed tasks S = {}
3  repeat
4        array L = unscheduled tasks that were not considered for team creation,
                   sorted by decreasing f(i,S)
```

```
5           counter = 0
6           repeat
7                   select task: i = L[counter]
8                   create Team for task i //according to the procedure described above
9                   if create team == possible
10                          update available technicians
11                          S = S ∪ {i}
12                          assign task i to new team
13                  else counter++
14          until counter >= |L| or create team == possible
15  until all tasks are scheduled or all technicians minus the safety stock χ|T| are
    planned
```

7.1.2 Greedy heuristic

After the team building phase, the unscheduled tasks are assigned to the existent teams by using the greedy approach from Section 6.1.2 with some extensions. Again, the principle is to calculate the best insertion cost for all tasks and the one whose insertion is associated with the lowest increase or the biggest decrease in the objective value is assigned to its lowest cost position. Nevertheless, since there are unscheduled technicians from the safety stock which can be used to reinforce the existent teams, the skill requirement constraints are ignored. So the task with the lowest feasible insertion cost is selected to be inserted (ignoring the skills). If the team is skilled sufficiently, the task is inserted. If not, the team is tried to be complemented by the free technicians by performing the *team completion algorithm* as described in depth in Section 7.2.3. If it is not possible to complement the team either, the task with the next lowest insertion cost is considered. To save computational time, a list of all feasible tasks with insertion costs and insertion positions is set up and only the values which are affected by an insertion are recalculated.

The process stops if no more tasks can be scheduled to the existent teams. Nevertheless, if there are still free technicians, the algorithm tries to set up a further team with the objective to schedule some of the remained tasks. The process is explained in detail in Section 7.2.4.

7.2 Solution improvement

The initial solution is improved by applying the ALNS as described above. Again, several sub-heuristics for the solution destruction and repair are available, which makes it possible for the algorithm to adjust itself to different problem specifications. The heuristic pairs (destroy-repair) to apply are selected by running *the roulette wheel selection* mechanism as in 5.3, whereas the probabilities for getting chosen depend on the heuristics´ previous performance. A new solution is obtained by applying the heuristics. Whether the new result is accepted or rejected, is decided by using the simulated annealing framework as for the version without team building.
The process of continuously destroying and repairing the schedule to create new solutions is repeated until the stopping criterion is met.

7.2.1 Destroy heuristics

Before starting the destruction of the current solution, again, the number of tasks to remove u is determined by drawing a random number from the interval of min $\{0.1n_a, 30\}$ and max $\{0.4n_a, 60\}$ as described in Section 6.2.1.
To destroy the solution, the destroy heuristics *random destroy*, *worst destroy*, *related destroy* and *cluster destroy* are applied in the same way as described for the solution method without team building. What's more, a fifth destroy method named *team destroy* is implemented.
To avoid needless waiting times, the solution is compressed after every removal by trying to serve the tasks scheduled to an affected team earlier. Furthermore, every affected team is checked for redundant technicians as in Cordeau et al. (2008). A technician is redundant for a team if all scheduled tasks could be executed without his assistance. As an extreme example, consider the case when all tasks are removed from a team´s schedule. Then, all technicians belonging to that team would be redundant. The basic idea is to set free unused technicians in order to use them for establishing new team configurations during the search. This enables to search the solution space at a much larger level. The unused technicians are identified by checking every team, which was affected by the task removal whether it could complete all its assigned tasks without one of its technicians. More formally, the algorithm checks for every technician in an affected team if any assigned task would be performable without him. Among all

redundant technicians, one is chosen randomly and is unscheduled. This process continues until no more technicians can be exempted.

7.1.2.1 Team Destroy

The team version method is extended by the *team destroy* heuristic, in which entire teams are deleted from the schedule. Teams are chosen randomly and removed until the number of removed tasks is equal or greater than u as outlined in Algorithm 6. The main idea of this method is the same as the checking for redundant technicians, namely to release some scheduled technicians. This makes it easier for the repair heuristics to regroup them into different teams with different resources where preferably few skills are wasted.

Algorithm: 6

```
1  input: feasible solution x
2  removed = 0
3  repeat
5          select team j to remove randomly
6          remove all scheduled tasks from team j
7          removed += number removed tasks
8          delete team and set assigned technicians free
9  until removed >= u
```

7.2.2 Repair heuristics

The ruined solution is composed by again applying the *greedy* heuristic, the *regret-q* heuristics in dependency of the number of teams and the *insertion* heuristic. Nevertheless, it is important to note that the application of the *team destroy* heuristic, as well as the check for redundant technicians might change the number of existing teams during the search, which can cause problems when applying the *regret-q* heuristics. Since the destroy and repair heuristics are chosen simultaneously (Section 5.3), it makes it inevitable to check if the number of teams has decreased after every destruction and if the actual number of teams is lower than q. If both cases are true, the weights ρ_{ab} for the

unusable heuristic pairs are set to zero, the new selection probabilities are calculated and a new repair heuristic for the already applied destroy heuristic is chosen by the roulette wheel selection. Afterwards, the weights and probabilities are set to their former values since the number of teams can again increase during the search.

Essentially, the repair methods work as described above, but the grouping of the technicians calls for an advancement. The available technicians gained from the check for redundant technicians, and especially from the team destroy heuristic, are used to support the existing teams if they are understaffed to perform tasks. Basically, the greedy heuristic tries to insert tasks according to their insertion costs. The regret heuristics apply the regret measure to identify good tasks and positions to insert. Finally, the insertion heuristic is based on the $c_2(i, h, i')$ measure. Nevertheless, the fact that the teams are expansible makes the check for the skill feasibility negligible. If the task, selected by any heuristic to be the most adequate for an insertion, fits into the determined team, it is scheduled just like that. If not, the team is tried to be adjusted by applying the *team completion algorithm* as described in the next section. If the completion is successful, the task is scheduled, but when it is not possible to form an adapted team, the heuristics consider the next best task until no more tasks can be inserted feasibly in the existent teams.
Furthermore, it might happen that even if the existent teams are skilled sufficiently, no task can be inserted due to the time window constraints. To react to such situations the algorithm tries to form new teams if there are unscheduled technicians left. This process is described in Section 7.2.4.

7.2.3 Team completion

As a consequence of the initial safety stock of technicians, of the characteristics of the team destroy heuristic and of the check for redundant technicians, there are consistently unscheduled technicians available. These technicians can be used to try different team configurations by adding them to teams that do not have sufficient skills to execute the tasks´ requirements. This switching of technicians during the search might result in big solution improvements, since the solution space is explored at a much larger level than if the team configuration would be fixed after the construction heuristic. This is especially true, as there is no possibility to change the teams during the day as described

in the problem formulation. Therefore, the algorithm must setup the teams in a way that they become as effective as possible. The *team completion algorithm* uses the same basic idea as the team construction method described in Section 7.1.1 and works for all repair and construction heuristics as follows.

First, the chosen repair or construction heuristic is applied to set up a list L composed of all feasible tasks and their corresponding insertion positions, but without considering the skill requirement constraints. The skills are ignored, since the aim of the team completion is to reinforce the group anyway if it is not arranged sufficiently. The list is sorted according to the applied heuristic: for the greedy heuristic in increasing order of the insertion costs and for the regret and insertion heuristics in decreasing order of the regret and $c_2(i, h, i')$ values respectively.

Now, the first task on the list L is tried to be scheduled at its position determined earlier. If the team is skilled sufficiently, the task is inserted and the algorithm goes on with the recalculation of the insertion costs and the updating of list L.

If it is not possible to serve the task with the team´s existing skills, but there are still unscheduled technicians, the algorithm tries to add some of these technicians to reinforce the team. The procedure is the same as for the team creation above. The task to insert can be imagined as the seed task for which a team must be constructed. Of course, only the uncovered skills denoted as $w'^{i}_{\alpha\beta}$ must be considered, so the requirements which can be executed by the existing team r (composed of technicians with skill matrices $v^{j}_{\alpha\beta}$) must be deducted from the task´s requirement matrix.

$$w'^{i}_{\alpha\beta} = \max\left\{0, w^{i}_{\alpha\beta} - \sum_{j \in T} x_{jr} v^{j}_{\alpha\beta}\right\}$$

Given the new matrix $w'^{i}_{\alpha\beta}$, the algorithm selects the free technician that can perform the most out of the task´s uncovered demands and adds him to the team. This process continues until the team achieves the abilities to serve the task. As in the example above, if there are technicians that cover the same amount of requirements, the technician with the lowest waste in skills is the one to be selected.

If the team completion was successful, the algorithm recalculates the insertion costs, updates list L and tries to insert the first task on the list. If it is not possible to complement the team and adjust the skills, because of the lack of skilled technicians, the algorithm goes on by trying to insert the next task on the sorted list L.

This process continues until no more tasks can be scheduled feasibly.

7.2.4 Setup of new teams

After a solution is destroyed, the repair heuristics attempt to schedule as much tasks to the available teams as possible. Nevertheless, the time window constraints can prevent the scheduling of tasks, even if the teams are highly skilled. The algorithm reacts to such situations by trying to form additional teams with the remaining free technicians.
First, the unscheduled technician with the lowest ID number is selected and declared as the first member of the new team. Then, unscheduled tasks are assigned to the new team by using the *insertion heuristic*. Since only the new team is considered for the scheduling, the sequential approach of this heuristic is especially suited. Again, tasks are inserted according to their $c_2(i, h, i')$ value, and if the team with the single technician is not able to execute them, he gets reinforcement by applying the completion algorithm from above. Of course, this is only possible if there are additional free technicians. The process of creating new teams is continued until no tasks can be inserted feasibly. If a new team is not able to execute any task, it is deleted and the assigned technicians are set free.

8. Further improvement

Ropke and Pisinger (2006b) suggested the application of some noise to the objective function, since the randomization in the repair phase leads to better results. Their suggestion is based on the claim that even though the search is guided by the randomized *simulated annealing* framework, the exploration of the neighborhood is done by the repair heuristics, so there should be a randomization at that level as well. This further randomization leads to a diversification of the found solutions, which in turn can help to identify better results.

The noise term is applied to the objective function when calculating the insertion costs in such a way that not always the best adequate task at the best position is selected for insertion. The noise term is a randomly chosen value, but it is bound by the maximum distance between two tasks times parameter ϑ, which controls the amount of noise. This

constraint is set to choose the noise parameter in dependency of the problem instance characteristics.

Every time when calculating a task´s insertion cost C, a random number in the interval $[-\vartheta \max_{i,i' \in N} d_{ii'}, \vartheta \max_{i,i' \in N} d_{ii'}]$ is chosen and added to C to obtain C´. Whether C or C´ is considered in the insertion phase, is decided on the base of the past performance of the two cost values by using the adaptive mechanism as described in Section 5.3.

9. Data Sets

Given that the problem discussed in this thesis, is a combination of the *vehicle routing problem* and the *technicians and interventions scheduling for telecommunications problem*, the test instances are generated out of Solomon´s VRPTW benchmark problems and the test instances which were provided by the organizers of the ROADEF 2007 challenge. From the VRPTW point of view, 12 different instance groups are used: random (R1, R2), clustered (C1, C2) and semi-clustered (RC1, RC2) geographical data. All instances have 100 customers and one depot, whereas the distances (= travel times) between them are given in Euclidean distances. Problem sets R1, C1, and RC1 have a short scheduling horizon, which allows only a few customers per route because of tighter constraints. On the other hand, problem sets R2, C2, and RC2 are characterized by a long scheduling horizon, where many customers can be scheduled per route. Furthermore, every problem set differs in the time window density, which is defined as the percentage of customers that have time windows at all. The mentioned sets are split into 50% and 100% time window density problems.

The Solomon instances were combined with the ROADEF data sets in a way that each one of the 100 customers (tasks) was reconfigured with randomly chosen skill requirements. What´s more, three different groups of requirement matrices are used, namely 5x4, 6x6 and 7x4. However, the data is slightly changed to cope with the actual problems characteristics. The number of columns in the matrices indicate the number of different domain types and the number of the rows the number of skill levels. Due to the fact that two algorithm versions, with team building and without, are tested, we distinguish between problem sets where every task must be performed by a single

technician and problem sets where technicians must be grouped together to come up with the required skills.
The outsourcing costs that accrue if a task cannot be scheduled are calculated as follows:

$$o_i = 200 + \text{difficulty}_i{}^{\delta}$$

The outsourcing cost depends on the difficulty of the task plus a fix value of 200. 200 is derived from the fact that tasks are distributed in a 100 x 100 coordinate system and with the depot at (35/35) for instances R1 and R2, the farthest pendulum tour would result in

$$\left\lceil \sqrt{(100-35)^2 + (100-35)^2} * 2 \right\rceil = 200$$

This increase is done, because the main objective of the solution method is to serve as many customers as possible. Consequently, tasks are only given off if there is no feasible way to serve them, and not if they are too far away. Weight $\delta > 1$ is included to make tasks that require more technicians more expensive. An exponential increase of the outsourcing costs with increasing difficulties seems to represent a real life situation the best, so δ is set to 1.5 for testing this heuristic.

Finally, we come up with: 3 (R, C, RC) x 2 (long and short planning horizon) x 2 (high and low time window density) x 3 (requirement matrices) x 2 (with and without team building) = 72 data sets.

The corresponding list of the available technicians which perform the tasks is also derived from the ROADEF data sets. Two types of technician sets are used. The first set, which contains sufficient technicians to serve all tasks, is used to test the algorithm´s routing efficiency. Additionally, a second set is created with a reduced number of technicians. This is done to simulate situations where it is impossible or very difficult to schedule all tasks. In such situations, besides good routing, it is essential to create reasonable teams in order to exploit all the technicians´ skills.
The number of the available technicians is listed in Table 1, however, it must be noted that technicians are mostly unique due to their different training.

	Team Version		No Team Version	
	Complete Set Of Technicians	Reduced Set Of Technicians	Complete Set Of Technicians	Reduced Set Of Technicians
C101_5x4	90	22	17	6
C103_5x4	90	22	17	6
C201_5x4	90	11	8	3
C203_5x4	90	11	8	3
R101_5x4	90	22	25	10
R103_5x4	90	22	25	10
R201_5x4	90	11	7	3
R203_5x4	90	11	7	3
RC101_5x4	90	22	22	9
RC103_5x4	90	22	22	9
RC201_5x4	90	11	9	3
RC203_5x4	90	11	9	3
C101_6x6	130	26	16	6
C103_6x6	130	26	16	6
C201_6x6	130	12	7	3
C203_6x6	130	12	7	3
R101_6x6	130	26	26	11
R103_6x6	130	26	26	11
R201_6x6	130	12	7	3
R203_6x6	130	12	7	3
RC101_6x6	130	26	24	10
RC103_6x6	130	26	24	10
RC201_6x6	130	12	8	3
RC203_6x6	130	12	8	3
C101_7x4	110	27	17	7
C103_7x4	110	27	17	7
C201_7x4	110	13	8	3
C203_7x4	110	13	8	3
R101_7x4	110	27	28	12
R103_7x4	110	27	28	12
R201_7x4	110	13	10	3
R203_7x4	110	13	10	3
RC101_7x4	110	27	23	10
RC103_7x4	110	27	23	10
RC201_7x4	110	13	9	3
RC203_7x4	110	13	9	3

Table 1: Number of available technicians

10. Computational experiments

The algorithm was implemented in C++ and run on an Intel® Pentium® D CPU 3.20 GHz computer with 2 CPUs sharing a memory of 4GB. For testing the developed algorithms, the 72 problem instances described above were used once with the complete set of technicians where it is possible to schedule all tasks, but also with a reduced set, in which the number of technicians is limited so it is much harder to assign tasks.

10.1 Parameter tuning

The present solution method uses a lot of parameters, which is rather undesirable, because all of them would have to be balanced against each other in order to find the optimal tuning. Nevertheless, during the test runs it turned out that the algorithms are very insusceptible against slight changes in the parameter tuning. This robustness is exploited by mainly applying the parameters determined in the related papers written by Cordeau et al. (2008) and Ropke and Pisinger (2006b). Therefore, only the parameters that are added for this problem are tuned. In the following, a short overview of all parameters is presented.

The parameters used for the simulated annealing criterion are w_T and the cooling rate c. The parameters used in the destroy heuristics are p_{worst} in the *worst destroy* and $p_{related}$, $w_{distance}$, w_{time} and $w_{similarity}$ in the *related destroy* heuristic. *Cluster* and *team destroy* methods are parameter free. In the repair phase, only the insertion heuristic uses parameters α_1, α_2 and λ. For the version with team building, further parameters are applied, namely w_β, w_γ and χ for the initial team creation. The weight adjustment algorithm uses parameters σ_1, σ_2, σ_3 and r. Finally, parameter ϑ determines the amount of noise.

Only parameters w_β, α_1, α_2 and $w_{similarity}$ were tuned for this algorithm by applying all problem instances with the complete and the reduced types of technician sets. Since parameter w_β is only used for the solution construction together with the greedy heuristic, the initial solutions generated with different parameter adjustments were tested. The solution quality in dependency of different w_β settings is posted in Table 2. For calibrating α_1 and α_2, the insertion heuristic was used as a construction method and the created (initial) solutions were compared in Table 3. In Table 4 the results obtained

by the ALNS in dependency of different $w_{similarity}$ settings is posted. The parameters were set by ignoring their interaction among each other by first fixing one parameter and then continuing with the next. The parameter setting that led to the best average result was chosen.

w_β	0.5	0.75	1	1.25	1.5	**1.75**	2
Team V. / complete set of technicians	2551.593	2516.42	2547.81	2533.83	2537.24	2509.61	2527.09
Team V. / reduced set of technicians	10170.44	9967.87	10014.2	10004.3	9920.75	9920.75	9983.6
Avg.	6361.01	6242.15	6281.00	6269.05	6229.00	**6215.18**	6255.34

Table 2: Initial solutions created by the greedy heuristic with different w_β parameters

α_1	α_2	Team V. / complete set of technicians	Team V. / reduced set of technicians	No Team V. / complete set of technicians	No Team V. / reduced set of technicians	Avg.
0	1	3504.98	11850.85	2646.90	6109.84	6028.14
0.1	0.9	3403.80	11669.54	2607.76	6165.43	5961.64
0.2	0.8	3379.67	11350.33	2564.33	6046.04	5835.09
0.3	0.7	3378.96	11291.99	2542.51	6139.72	5838.29
0.4	0.6	3268.08	11082.22	2562.61	6164.49	5769.35
0.5	0.5	3286.63	10780.96	2557.91	6121.92	5686.85
0.6	0.4	3284.95	10840.57	2526.56	6061.93	5678.51
0.7	0.3	3238.53	10735.06	2530.10	6077.16	5645.21
0.8	0.2	3228.08	10730.20	2497.74	5949.82	5601.46
0.9	0.1	3231.50	10598.19	2483.69	6052.46	5591.46
1	**0**	3187.21	10502.18	2430.74	5808.48	**5482.15**

Table 3: Initial solutions created by the insertion heuristic with different α parameters

$w_{similarity}$	2	3	**4**	5	6
Team V. / complete set of techn.	1077.78	1076.79	1077.16	1076.75	1077.39
Team V. / reduced set of techn.	6141.13	6209.09	6071.73	6165.11	6179.16
No Team V. /complete set of techn.	1296.41	1296.83	1299.71	1297.33	1296.93
No Team V. / reduced set of techn.	3499.42	3509.91	3502.81	3505.79	3511.03
Avg.	3003.69	3023.15	**2987.85**	3011.24	3016.13

Table 4: Solutions created by the ALNS with different $w_{similarity}$ parameters

All used parameter settings are indicated in the parameter vector ($w_{distance}$, w_{time}, $w_{similarity}$, $p_{related}$, p_{worst}, w_T, $\alpha 1$, $\alpha 2$, λ, c, $\sigma 1$, $\sigma 2$, $\sigma 3$, r, ϑ, w_β, w_γ, χ) = (9, 3, 5, 6, 3, 0.05, 1, 0, 2, 0.99975, 33, 9, 13, 0.1, 0.025, 1.75, 1, 0.2).

10.2 Experimental results

In this section, the obtained results of the approaches described in Chapter 6 and 7 are presented. At the beginning the performance of the repair sub-heuristics is shown in Table 5 - 8. However, to make the methods comparable each of them was applied as a construction heuristic to use them to build a solution from scratch. The tables contain the averaged results of all problem sets described above. Each run was executed once since the heuristics are not affected by random parameters.

Table 5 and Table 6 show the results for the version without team building and Table 7 and 8 for the version with team building. Note that all construction heuristics for the version with team building use the same initial team construction procedure as described in Section 7.1.1.

Furthermore, for the results in Table 5 and 7 the entire set of technicians was used, and in Table 6 and 8 the reduced set. The columns of the tables stand for the different sub-heuristics that are compared to each other. These are also compared to the average result obtained by the ALNS. The rows indicate the average objective value across all instances of the respective data set and the corresponding number of tasks that could be scheduled, the number of technicians used and the time needed to obtain the result. Furthermore, in Table 7 and 8 the number of composed teams is indicated. For Table 5 and 6 this is not necessary, since all technicians work alone.

It is obvious that the results obtained by these simple construction heuristics are far away from those found by the ALNS, nevertheless it must be pointed out that the ALNS is based on them. Based on Tables 5 – 8, it can be said that each heuristic has different pros and cons for different problem instances.

In most of the cases the *regret-q* heuristics prove to have the best quality. However, in Table 8 the *insertion* heuristic outperforms all of the *regret-q* heuristics. The *greedy* heuristic achieves the best average solution in Table 8. Even though it is outperformed by the *regret-q* heuristics in the other instances, it has quite a stable performance.

	ALNS	Greedy	Insertion	Regret-2	Regret-3	Regret-4	Regret-m
Objective	1299.71	2205.68	2430.74	2028.84	1890.17	1831.88	1945.19
Assined Tasks	100.00	99.11	99.58	99.81	99.92	99.94	100.00
Assigned Technicians	10.77	12.39	13.28	12.42	12.31	11.94	11.39
Time (s)	78.94	0.00	0.00	0.02	0.02	0.02	0.03

Table 5: Construction heuristics (no team version / complete set of technicians)

	ALNS	Greedy	Insertion	Regret-2	Regret-3	Regret-4	Regret-m
Objective	3502.81	5602.19	5808.48	5040.68	5036.52	5076.09	5165.68
Assined Tasks	89.96	80.56	80.53	83.83	84.11	84.06	83.61
Assigned Technicians	7.56	7.56	7.64	7.58	7.53	7.58	7.61
Time (s)	53.04	0.00	0.00	0.01	0.01	0.01	0.01

Table 6: Construction heuristics (no team version / reduced set of technicians)

	ALNS	Greedy	Insertion	Regret-2	Regret-3	Regret-4	Regret-m
Objective	1077.16	2509.61	3187.21	2370.65	2393.43	2379.71	2379.86
Assined Tasks	100.00	98.86	98.78	99.31	99.19	99.25	99.44
Used Teams	8.73	31.69	31.72	31.72	31.69	31.67	31.78
Assigned Technicians	41.66	104.64	99.81	105.06	104.78	104.92	104.83
Time (s)	84.63	0.02	0.02	0.07	0.07	0.06	0.07

Table 7: Construction heuristics (team version / complete set of technicians)

	ALNS	Greedy	Insertion	Regret-2	Regret-3	Regret-4	Regret-m
Objective	6071.73	9920.75	10502.18	11276.23	10954.81	10872.12	11224.79
Assined Tasks	79.44	60.61	58.92	57.06	57.47	58.08	57.11
Used Teams	6.19	4.86	5.22	5.14	5.11	5.08	5.11
Assigned Technicians	18.47	18.11	18.39	18.39	18.44	18.36	18.42
Time (s)	72.49	0.00	0.00	0.01	0.01	0.01	0.01

Table 8: Construction heuristics (team version / reduced set of technicians)

To be able to compare the developed approach to already existent ones, the ALNS algorithm was applied to the classical VRPTW instances of Solomon. The solution method was adapted in a way that it fits to the problem sets characteristics. This implies the neglecting of the skill requirements and outsourcing costs but the introduction of a team (vehicle) capacity constraint.

The most heuristics applied to the VRPTW focus on the minimization of the route numbers and only as a second objective try to reduce the routing costs. Therefore, the ALNS is compared to the exact solutions where the number of routes is irrelevant but the routing costs are considered. As a consequence, the route number was not bound either when applying the ALNS.

Table 9 demonstrates the averaged solutions over five runs, each of them using 25000 iterations of the ALNS algorithm. The table reports the basic VRPTW instances as described in Section 9 (column 1), the exact (optimal) solution as posted in Kallehauge et al. (2006) (column 2), the travel distance achieved by the ALNS and the gap between the obtained solution and the best solution (column 3 and 4). The average number of routes and the average time for one run are listed in column 5 and 6. In the last row the averages of the columns are indicated.

The average gap of 0.50% verifies the goodness of the solution finding ability of the ALNS.

		ALNS			
Instance	Best	Avg.	Gap (%)	Number Of Routes	Time (s)
C101	827.30	828.94	0.20%	10	53.08
C103	826.30	828.07	0.21%	10	96.39
C201	589.10	591.56	0.42%	3	84.83
C203	588.70	591.17	0.42%	3	152.48
R101	1637.70	1643.29	0.34%	20	76.92
R103	1208.70	1213.64	0.41%	14	95.17
R201	1143.20	1150.25	0.62%	8.2	102.98
R203		876.45		6	146.82
RC101	1619.80	1635.17	0.95%	15.6	78.26
RC103		1276.04		11.2	88.99
RC201	1261.80	1274.10	0.97%	9	100.77
RC203		939.06		5	129.80
Avg.	1078.07	1070.64	0.50%	9.58	100.54

Table 9: ALNS applied to VRPTW vs. optimal solution values

The results for the *service technician routing and scheduling problem* obtained by the ALNS are demonstrated in Tables 10 - 13. The results are obtained by averaging five runs of the heuristic with 25000 iterations. Tables 10 and 11 show the outcome of the no team version, where the complete set of technicians is used for Table 10 and the

reduced set for Table 11. In Tables 12 and 13, the findings of the team version are listed. Table 12 shows the results with the complete set of technicians and Table 13 with the reduced set. The tables contain the average objective value (column 2) and the worst and the best found solution during five runs (columns 3 and 4). In Tables 10 and 11, columns 5 and 6 report how many tasks could be assigned to how many technicians, whereas in Tables 12 and 13, columns 5, 6 and 7 indicate how many tasks could be performed by how many teams and the number of technicians belonging to the team. The last column indicates the average computational time in seconds. The last row again shows the average values of the columns.

Unfortunately, the results cannot be compared to other heuristics, since this is the first time that the mentioned problem sets were applied. To have an idea about the quality of the solution, the instances in which all tasks are scheduled can be compared to the VRPTW solutions.

The average result of the team version algorithm which uses a large set of technicians is very close to the best routing of the VRPTW. Likewise, the heuristic performed quite well for the no team version method in which there is no possibility to change the teams´ skill configurations.

Instance	Objective	MAX	MIN	Assigned Tasks	Assigned Technicians	Time (s)
C101_5x4_noTeam	1117.85	1156.18	1097.67	100	12.8	65.96
C103_5x4_noTeam	1031.31	1043.10	1015.46	100	11.8	84.71
C201_5x4_noTeam	1165.95	1183.22	1158.97	100	7.2	54.06
C203_5x4_noTeam	1063.25	1084.87	1046.93	100	5	87.70
R101_5x4_noTeam	1680.20	1688.34	1675.35	100	20	84.40
R103_5x4_noTeam	1250.88	1260.45	1240.19	100	14.8	103.07
R201_5x4_noTeam	1455.05	1477.68	1447.32	100	6.8	56.44
R203_5x4_noTeam	1107.79	1115.58	1099.31	100	6	85.69
RC101_5x4_noTeam	1703.03	1717.75	1686.36	100	16.8	77.71
RC103_5x4_noTeam	1372.35	1392.17	1346.51	100	12.4	91.52
RC201_5x4_noTeam	1617.57	1635.67	1607.71	100	8.4	58.40
RC203_5x4_noTeam	1161.53	1161.53	1161.53	100	6	86.38
C101_6x6_noTeam	1016.62	1065.60	989.21	100	11	68.04
C103_6x6_noTeam	910.03	929.59	893.94	100	10.6	87.41
C201_6x6_noTeam	821.55	821.55	821.55	100	4	56.32
C203_6x6_noTeam	690.53	691.93	689.60	100	4	100.55
R101_6x6_noTeam	1667.52	1675.72	1662.69	100	19.6	94.96
R103_6x6_noTeam	1222.46	1226.02	1218.23	100	14	114.40
R201_6x6_noTeam	1277.36	1287.53	1268.95	100	6	62.30
R203_6x6_noTeam	949.73	956.83	934.98	100	5	97.80
RC101_6x6_noTeam	1693.50	1703.37	1686.29	100	16	90.17
RC103_6x6_noTeam	1320.30	1336.70	1283.74	100	11.8	103.76
RC201_6x6_noTeam	1408.52	1443.51	1394.40	100	6.2	61.73
RC203_6x6_noTeam	1027.80	1044.58	1006.25	100	5	90.81
C101_7x4_noTeam	1394.70	1424.97	1365.49	100	15.4	58.01
C103_7x4_noTeam	1241.41	1249.63	1233.40	100	12.8	75.76
C201_7x4_noTeam	1274.84	1302.56	1256.30	100	8	49.83
C203_7x4_noTeam	1151.49	1152.94	1150.85	100	7.8	77.20
R101_7x4_noTeam	1790.33	1805.71	1783.35	100	21.6	87.75
R103_7x4_noTeam	1382.22	1405.34	1359.45	100	16.4	101.53
R201_7x4_noTeam	1407.66	1416.72	1398.14	100	8.6	58.61
R203_7x4_noTeam	1168.13	1169.27	1164.69	100	8	80.91
RC101_7x4_noTeam	1846.80	1892.21	1800.65	100	18.2	75.20
RC103_7x4_noTeam	1448.71	1474.43	1431.92	100	13.6	84.99
RC201_7x4_noTeam	1708.08	1718.79	1697.82	100	9	53.65
RC203_7x4_noTeam	1242.39	1248.79	1235.75	100	7.2	74.20
Avg.	1299.71	1315.58	1286.42	100.00	10.77	78.94

Table 10: ALNS (no team version / complete set of technicians)

Instance	Objective	MAX	MIN	Assigned Tasks	Assigned Technicians	Time (s)
C101_5x4_noTeam	5716.57	5806.38	5656.63	76.6	8	35.34
C103_5x4_noTeam	2746.88	2888.93	2600.27	92.6	8	50.87
C201_5x4_noTeam	2755.52	2755.52	2755.52	94	4	36.56
C203_5x4_noTeam	2392.62	2393.62	2389.21	94	4	70.15
R101_5x4_noTeam	5859.14	5979.65	5761.62	77.6	12	53.58
R103_5x4_noTeam	1797.26	1957.80	1707.51	98	12	61.46
R201_5x4_noTeam	2847.96	2885.80	2838.50	94	4	38.86
R203_5x4_noTeam	2332.23	2332.23	2332.23	94	4	68.24
RC101_5x4_noTeam	5201.64	5311.38	5080.75	81.4	11	51.01
RC103_5x4_noTeam	2365.39	2492.17	2173.86	95.8	11	55.39
RC201_5x4_noTeam	3092.29	3093.34	3088.23	94	5	40.76
RC203_5x4_noTeam	2545.58	2551.54	2543.86	94	5	69.11
C101_6x6_noTeam	7734.60	7776.46	7714.11	68	8	40.31
C103_6x6_noTeam	5126.26	5169.75	4969.91	81.2	8	55.24
C201_6x6_noTeam	3289.97	3311.21	3278.07	90.4	4	38.74
C203_6x6_noTeam	2461.28	2472.21	2449.63	94	3	66.93
R101_6x6_noTeam	6175.71	6319.02	5959.53	76.8	13	60.40
R103_6x6_noTeam	2313.35	2356.46	2263.68	95.4	12	72.20
R201_6x6_noTeam	3557.53	3604.69	3545.74	91	4	38.60
R203_6x6_noTeam	2437.51	2438.44	2437.28	94	3	71.98
RC101_6x6_noTeam	5465.28	5748.17	5165.14	81.2	12	59.87
RC103_6x6_noTeam	2270.25	2542.33	2132.54	97.4	12	65.06
RC201_6x6_noTeam	4566.26	4742.02	4422.86	87.4	4	34.49
RC203_6x6_noTeam	2667.29	2668.47	2663.33	94	3	61.74
C101_7x4_noTeam	5251.51	5297.75	5208.30	81	9	38.58
C103_7x4_noTeam	2039.42	2169.19	1968.95	97.8	9	50.30
C201_7x4_noTeam	2773.41	2773.41	2773.41	95	4	32.77
C203_7x4_noTeam	2281.59	2284.92	2277.79	95	4	60.74
R101_7x4_noTeam	5290.47	5403.91	5144.04	82	14	47.88
R103_7x4_noTeam	2258.08	2339.01	2151.79	96.2	14	63.91
R201_7x4_noTeam	2679.13	2700.10	2668.81	95	5	41.82
R203_7x4_noTeam	2217.18	2230.42	2199.10	95	5	66.84
RC101_7x4_noTeam	5693.87	5857.17	5520.77	80	12	53.15
RC103_7x4_noTeam	2682.16	3066.69	2586.03	94.6	12	58.55
RC201_7x4_noTeam	2930.85	2942.84	2908.33	95	5	39.73
RC203_7x4_noTeam	2285.18	2303.44	2277.62	95	5	58.21
Avg.	3502.81	3582.40	3433.75	89.96	7.56	53.04

Table 11: ALNS (no team version / reduced set of technicians)

Instance	Objective	MAX	MIN	Assigned Tasks	Used Teams	Assigned Technicians	Time (s)
C101_5x4_Team	828.94	828.94	828.94	100	10	50	56.66
C103_5x4_Team	815.44	815.44	815.44	100	10	51.6	99.48
C201_5x4_Team	591.56	591.56	591.56	100	3	25.8	44.93
C203_5x4_Team	591.17	591.17	591.17	100	3	25.6	71.09
R101_5x4_Team	1648.76	1653.99	1642.88	100	19.4	84.8	88.72
R103_5x4_Team	1215.32	1218.14	1213.62	100	14	63.6	104.72
R201_5x4_Team	1191.13	1198.41	1179.14	100	5.2	32.6	71.72
R203_5x4_Team	896.97	901.43	892.92	100	4.2	29	111.33
RC101_5x4_Team	1629.60	1644.25	1618.14	100	15.2	72	76.02
RC103_5x4_Team	1261.76	1272.62	1249.37	100	11.2	56.6	81.46
RC201_5x4_Team	1316.82	1329.95	1307.20	100	5.4	34.8	69.00
RC203_5x4_Team	946.55	951.06	940.81	100	4.4	31.2	101.51
C101_6x6_Team	828.94	828.94	828.94	100	10	29.2	63.81
C103_6x6_Team	815.44	815.44	815.44	100	10	31.2	100.44
C201_6x6_Team	591.56	591.56	591.56	100	3	10.2	46.60
C203_6x6_Team	591.17	591.17	591.17	100	3	10	89.67
R101_6x6_Team	1649.42	1651.01	1644.97	100	19.6	50.2	106.09
R103_6x6_Team	1215.59	1218.53	1213.62	100	14	37.6	108.51
R201_6x6_Team	1195.38	1198.41	1189.13	100	5	16.6	71.88
R203_6x6_Team	897.83	904.47	892.92	100	4	13.2	114.51
RC101_6x6_Team	1622.77	1629.79	1618.14	100	15	42.4	86.49
RC103_6x6_Team	1257.24	1266.84	1250.13	100	11	31	87.64
RC201_6x6_Team	1305.46	1317.46	1298.92	100	5.6	18.4	73.43
RC203_6x6_Team	948.29	951.06	944.80	100	4.2	14.6	101.57
C101_7x4_Team	828.94	828.94	828.94	100	10	59.6	64.76
C103_7x4_Team	815.44	815.44	815.44	100	10	59	102.26
C201_7x4_Team	591.56	591.56	591.56	100	3	27.4	47.78
C203_7x4_Team	591.17	591.17	591.17	100	3	27.6	79.72
R101_7x4_Team	1647.54	1651.07	1642.88	100	19.8	92.2	94.63
R103_7x4_Team	1215.53	1217.60	1213.62	100	14	74.6	108.46
R201_7x4_Team	1189.79	1200.95	1171.30	100	5.2	39	73.11
R203_7x4_Team	895.23	895.23	895.23	100	4	35.6	112.77
RC101_7x4_Team	1627.74	1641.16	1618.14	100	15.2	78.8	77.28
RC103_7x4_Team	1265.18	1276.18	1259.13	100	11	67.2	85.71
RC201_7x4_Team	1311.73	1325.91	1299.16	100	5.6	41.2	72.07
RC203_7x4_Team	944.73	950.01	937.45	100	4.2	35.2	100.78
Avg.	1077.16	1081.86	1072.64	100.00	8.73	41.66	84.63

Table 12: ALNS (team version / complete set of technicians)

Instance	Objective	MAX	MIN	Assigned Tasks	Used Teams	Assigned Technicians	Time (s)
C101_5x4_Team	8873.86	9384.44	7414.42	62.6	6.4	22	60.38
C103_5x4_Team	6516.33	7329.40	3943.93	73.6	6.4	22	57.36
C201_5x4_Team	4879.62	5756.14	4326.97	84.2	3	11	62.73
C203_5x4_Team	4275.89	7357.72	3464.24	85.4	2.8	11	93.71
R101_5x4_Team	11415.84	12383.40	10335.50	51.2	7	22	31.76
R103_5x4_Team	9169.39	9927.13	8553.07	60	5.6	22	41.98
R201_5x4_Team	5723.10	6575.80	4395.26	80.2	2.4	11	35.34
R203_5x4_Team	2410.72	2727.62	2206.64	95	2.8	11	99.03
RC101_5x4_Team	9393.39	10481.10	8648.72	61.4	7.6	22	47.06
RC103_5x4_Team	8342.85	9767.47	5809.87	65.4	6.4	22	49.62
RC201_5x4_Team	6468.31	7842.63	5721.41	78.6	2.8	11	49.58
RC203_5x4_Team	2402.10	2445.12	2287.43	96.2	3	11	82.97
C101_6x6_Team	2587.69	2822.44	2439.61	95.8	13	26	94.62
C103_6x6_Team	1245.10	1349.32	1142.06	99.6	10.8	24.8	102.64
C201_6x6_Team	2314.92	2483.80	2185.09	96	4.8	12	79.60
C203_6x6_Team	2111.98	2859.82	1628.75	96	5.6	12	131.42
R101_6x6_Team	6920.80	7334.27	6640.74	77.8	14.2	26	89.70
R103_6x6_Team	3007.50	3078.38	2970.66	94	13	26	109.46
R201_6x6_Team	2685.26	2858.94	2489.11	95.8	4.8	12	85.10
R203_6x6_Team	1498.98	1913.47	1384.22	98.6	4.8	12	117.72
RC101_6x6_Team	5190.32	5596.93	4847.97	86.4	13.8	26	115.91
RC103_6x6_Team	2949.12	3076.42	2826.37	94.6	12.2	26	100.69
RC201_6x6_Team	2976.32	3127.04	2835.16	95.6	5.6	12	84.16
RC203_6x6_Team	1955.94	2364.50	1553.31	97.8	4.6	12	116.40
C101_7x4_Team	8507.47	9847.93	7700.61	70.2	7.6	27	54.92
C103_7x4_Team	5947.76	6422.18	5420.87	80.2	7.2	27	73.37
C201_7x4_Team	9436.25	9713.15	8328.67	65.6	2.2	13	22.09
C203_7x4_Team	8551.38	8552.59	8550.46	67	2	13	38.13
R101_7x4_Team	12215.94	12645.00	11741.70	55.6	9	27	53.77
R103_7x4_Team	8442.06	9736.15	7757.92	69.6	7.4	27	58.46
R201_7x4_Team	8423.12	8610.98	8242.72	71.4	2	13	41.17
R203_7x4_Team	5069.80	5314.86	4301.80	86	2.2	13	81.81
RC101_7x4_Team	11006.04	11436.80	10793.60	60.2	8.6	27	79.26
RC103_7x4_Team	8817.29	9478.41	8069.80	69	7.4	27	70.13
RC201_7x4_Team	10350.58	10438.20	10280.60	63.6	2	13	25.11
RC203_7x4_Team	6499.11	6649.61	6426.90	79.8	2	13	72.55
Avg.	6071.73	6658.03	5490.73	79.44	6.19	18.47	72.49

Table 13: ALNS (team version / reduced set of technicians)

10. Conclusion

The *adaptive large neighborhood search* proposed by Ropke and Pisinger (2006b), Pisinger and Ropke (2007) and Cordeau et al. (2008) was used to solve the problem described in this thesis. The method was adapted according to better fit the problem characteristics. The differences appear in the way how the initial solution is created, but there are also changes in the repair and destroy heuristics and the way they are selected.
In contrast to Pisinger and Ropke (2007) who created the initial solution by using the regret-2 heuristic, the current method uses the greedy heuristic. The reason for this is an existing possibility that only one route is needed to serve all customers.
Another adaptation is that the original method doesn´t use the insertion heuristic to repair the destroyed solutions. Additionally, in this thesis the selection of the sub-heuristics is done by recording the joint performance of pairs of destroy-repair methods, whereas they are chosen independently in the underlying solution methods.

To test the developed algorithms 72 problem instances where created by combining Solomon´s VRPTW instances and the instances from the ROADEF 2007 challenge. Furthermore, each instance was tested by applying an oversized and a reduced set of technicians.
Unfortunately, there is no comparison to the application of the ALNS to the current problem sets. Nevertheless, it is assumed that it is going to be able to compete with upcoming solution methods, as it performed very well when solving the VRPTW. Additionally, the results of the STRSP instances with the complete set of technicians are close to the VRPTW solutions.

Further extensions of the solution method could consist of elements, which would make it even a better fit for real life situations. Such extensions could include the integration of lunch breaks, overtime of the technicians or the assignment of trainees to the teams.

Bibliography

Ahuja R. K., Ergun Ö., Orlin J. B., Punnen A. P.: A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics, 2002, 123: 72–102.

Bräysy O., Gendreau M.: Vehicle routing problem with time windows, part I: route construction and local search algorithm. Transportation Science, 2005, 39:104–118.

Bräysy O., Gendreau M.: Vehicle routing problem with time windows, part II: metaheuristics. Transportation Science, 2005, 39: 119–139.

Cordeau J.F., Laporte G., Pasin F., Ropke S.: Scheduling Technicians and Tasks in a Telecommunications Company. Les Cahiers du GERAD, 2008, G–2008–45.

Dutot P.-F., Laugier A., Bustos A.-M.: Technicians and interventions scheduling for telecommunications. France Telecom R&D, 2006.

http://web.cba.neu.edu/~msolomon/problems.htm (09.11.2009)

http://www.g-scop.inpg.fr/ChallengeROADEF2007/index.php?page=4&lang=1 (09.11.2009)

Kirkpatrick S., Gelatt C.D. Jr., Vecchi M. P.: Optimization by simulated annealing. Science, 1983, 220: 671–680.

Kallehauge B., Larsen J., Madsen O. B. G.: Lagrangian duality applied to the vehicle routing problem with time windows. Computers & Operations Research, 2006, 33: 1464–1487.

Kruskal J. B. Jr.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society, 1956, 7: 48- 50.

Mladenović N., Hansen E.: Variable neighborhood search, Computers & Operations Research,1997, 24 (11): 1097-1100.

Potvin J.-Y., Rousseau J.-M.: A parallel route building algorithm for the vehicle routing problem with time windows. European Journal of Operational Research, 1993, 66: 331-340.

Pisinger D., Ropke S.: A general heuristic for vehicle routing problems. Computers & Operations Research, 2007, 34: 2403–2435.

Ropke S., Pisinger D.: A unified heuristic for a large class of Vehicle Routing Problems with Backhauls. European Journal of Operational Research, 2006a, 171: 750–775.

Ropke S., Pisinger D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 2006b, 40: 455–472.

Schrimpf G, Schneider J, Stamm-Wilbrandt H., Dueck G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics, 2000, 159: 139–171.

Shaw P.: A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, Scotland, 1997.

Shaw P.: Using constraint programming and local search methods to solve vehicle routing problems. CP-98, Fourth international conference on principles and practice of constraint programming, Lecture notes in computer science, 1998, 1520: 417–431.

Solomon M. M.:. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 1987, 35: 254–265.

Xu J., Chiu S. Y.: Effective heuristic procedures for field technician scheduling problem, Journal of Heuristics, 2001, 7: 495-509.

Abstract

The current thesis deals with a real life planning challenge faced by many companies, namely the *service technician routing and scheduling problem*. The problem appears when a given pool of differently skilled technicians has to execute a set of tasks which have different skill requirements. Furthermore, tasks are located at different sites and their visiting is constrained by time windows. As in some cases the tasks´ requirements are too high to be met by a single technician, it may also be necessary to group some technicians together to aggregate their skills.

Basically, this problem can be interpreted as an extended vehicle routing problem (VRP) in which the vehicles are replaced by the different teams or technicians, and the customers by tasks with diverse demands. The challenge is now to create routes that visit as many tasks as possible, by considering the demanded skills and the time window constraints.

To tackle this problem the *adaptive large neighborhood search* heuristic (ALNS) was applied. It proved to have a very good performance when used for solving the classic VRPs in which the routing aspect is in the front. Nevertheless, when applied to the *technician and task scheduling problem*, in which the routing is not an issue but the team creation and assignment, the ALNS achieved impressive results as well. It is based on the idea of continuously destroying the current solution and rebuilding it in an improved way. The destruction is done by removing some of the already scheduled tasks and the repair by reinserting them at a position where they cause less costs.

In contrast to the underlying *large neighborhood search* heuristic, the ALNS uses several destroy and repair sub-heuristics, to allow the algorithm to adjust itself to different problem types (therefore the name). To decide which sub-heuristics to use, the past performance of each destroy-repair method pair is recorded. The better a pair of methods performs, the higher the probability that it is applied again.

Two solution approaches were created for two variants of this problem. First, when technicians work alone, since all tasks can be executed by an individual, sufficiently skilled technician. And second, when it is impossible to perform some tasks alone, so several technicians must be grouped together in order to form teams.

Zusammenfassung

Die vorliegende Arbeit befasst sich mit einer Planungsherausforderung, der sich viele Unternehmen täglich stellen müssen, nämlich dem *service technician routing and scheduling* Problem. Das Problem entsteht dadurch, dass eine fixe Belegschaft von unterschiedlich ausgebildeten Technikern eine Menge von Aufträgen bearbeiten muss die unterschiedliche Fachkenntnisse erfordern. Erschwerend kommt hinzu, dass die Aufträge örtlich unterschiedlich gelegen sind und nur in bestimmten Zeitfenstern besucht werden können. Darüber hinaus kommt es vor, dass manche Aufträge zu hohe Anforderungen an einzelne Techniker stellen. Dies erfordert die Zusammengruppierung von mehreren Technikern um mit den vereinten Kenntnissen den nachgefragten Anforderungen gerecht zu werden.

Grundsätzlich kann dieses Problem als eine Erweiterung des *vehicle routing* Problems (VRP) verstanden werden, wobei die Fahrzeuge durch die Techniker oder Teams ersetzt werden und die Kunden durch die Aufträge mit den unterschiedlichen Anforderungen. Die Herausforderung besteht nun darin, Routen zu generieren die möglichst viele Aufträge einschließen, wobei den nachgefragten Fähigkeiten und den Zeitfenstern entsprochen wird.

Um das Problem zu lösen wird die sogenannte *adaptive large neighborhood search* Heuristik (ALNS) angewendet. Bei der Anwendung an klassischen VRPs, bei denen der Routenplanungsaspekt im Vordergrund steht, konnte diese Methode sehr gute Ergebnisse erzielen. Aber auch bei der Anwendung an dem *technician and task scheduling* Problem, bei dem keine Routen geplant werden, aber die Zusammenstellung und Einteilung von unterschiedlichen Teams gefordert wird, konnte ALNS seine Stärken unter Beweis stellen. Es basiert auf der Idee, eine aktuelle Lösung fortlaufend zu zerstören um sie anschließend besser wiederherzustellen. Das Zerstören wird durch entfernen bereits eingeplanter Aufträge bewirkt und die Reparatur durch das wiedereinfügen in Positionen in denen sie weniger Kosten verursachen. Im Gegensatz zu der zu Grunde liegenden *large neighborhood search* Heuristik, werden bei der ALNS Methode mehrere Zerstörungs- und Reparatur-Subheuristiken verwendet. Dies ermöglicht es dem Algorithmus sich an unterschiedliche Probleminstanzen anzupassen (daher auch der Name). Die Entscheidung, welche Subheuristiken verwendet werden basiert auf dem Erfolg, den die Zerstörungs-Reparatur Heuristikpaare in vorangegangenen Iterationen erzielen konnten. Je besser die erreichten Lösungen eines

Methodenpaares, desto höher die Wahrscheinlichkeit, dass es noch einmal zum Einsatz kommt.

Ein Lösungsansatz wurde für beide Varianten des Problems entwickelt. Erstens, bei dem alle Techniker alleine arbeiten, da alle Aufträge individuell bearbeitet werden können. Zweitens, bei dem es nicht möglich ist manche Aufträge alleine zu bewältigen und deswegen Teams zusammengestellt werden müssen.

Curriculum vitae

Personal data:

Name:	Attila Kovacs
Date of birth:	September 14, 1983
Citizenship:	Austria

Education:

since 2003	University of Vienna International Business Administration Financial Services and Logistics Thesis: “Heuristics for Service Technician Routing and Scheduling Problems”
1997 – 2002	Secondary technical college of Eisenstadt Aeronautical engineering Thesis: “Test bench for a model turbo jet engine”

International experience:

09/2008 – 02/2009	University of Las Palmas de Gran Canaria Erasmus student exchange
June 2001	Aviation practical training JAR 66, Budapest EU-Technical training program LEONARDO da VINCI

Professional experience:

10/2008 - 01/2008 and 03/2009-07/2010	Study assistant University of Vienna

	Chair for Production and Operations Management
Since 06/2008	Driving instructor
	Driving school “Fahrschule Ing. Kovacs”
10/2005 – 07/2008	Educator
	Secondary school “Institut Neulandschule”

Languages:

Hungarian	native
German	native
English	fluent
Spanish	good