Workforce Routing and Scheduling for Electricity Network Maintenance with Downtime Minimization

Asvin Goel[a], Frank Meisel[b,*]

[a]*School of Engineering and Science, Jacobs University, Bremen, Germany*
[b]*School of Economics and Business, Martin-Luther-University Halle-Wittenberg, Germany*

Abstract

We investigate a combined routing and scheduling problem for the maintenance of electricity networks. In electricity networks power lines must be regularly maintained to ensure a high quality of service. For safety reasons a power line must be physically disconnected from the network before maintenance work can be performed. After completing maintenance work the power line must be reconnected. Each maintenance job therefore consists of multiple tasks which must be performed at different locations in the network. The goal is to assign each task to a worker and to determine a schedule such that the downtimes of power lines and the travel effort of workers are minimized. For solving this problem, we combine a Large Neighborhood Search meta-heuristic with mathematical programming techniques. The method is evaluated on a large set of test instances which are derived from network data of a German electricity provider.

Keywords: OR in Energy, Maintenance, Downtime Minimization, Routing, Scheduling

1. Introduction

Availability of electric energy is a crucial factor of today's industrialized economies and of the living standard of every modern society. In order to provide electricity to the customers, dense networks of power lines and transformer stations are established for connecting power plants to businesses and households. The reliability of such networks is typically assessed by the System Average Interruption Index (SAIDI). This index gives the average outage duration faced by customers within a year. For example, the SAIDI-index for customers in Germany was 14.9 minutes in

*Corresponding author, Tel.: +49 345 5523 383; Fax: +49 345 5527 198; Gr. Steinstr. 73, 06108 Halle, Germany; E-mail addresses: a.goel@jacobs-university.de (A. Goel), frank.meisel@wiwi.uni-halle.de (F. Meisel).

2010, corresponding to a 99.998% availability rate, see Bundesnetzagentur (2012). Clearly, maintaining the power utilities is one of the keys for achieving such a reliable supply of electric energy.

A particular feature of electricity network maintenance is that a maintenance job can comprise several interdependent tasks that occur at geographically dispersed locations. This is because the affected part of a network has to be disconnected from the rest of the network before maintenance work can be done. Figure 1 gives an example of such task-interdependencies. Here, the power line that connects transformer stations TS1 and TS2 requires maintenance. For safety reasons, the power line must first be turned off and physically disconnected from the network at both transformer stations. Afterwards, the actual maintenance can be performed. Then, two more tasks have to be performed at the transformer stations for bringing the power line back into service. Obviously, the affected power line is unavailable from the begin of the first involved task until the completion of the last involved task. The overall downtime of the asset can be minimized by simultaneously disconnecting the power line at stations TS1 and TS2, performing the actual maintenance task immediately after the power line is disconnected, and reconnecting the power line immediately thereafter. As the disconnecting, reconnecting, and the actual maintenance task

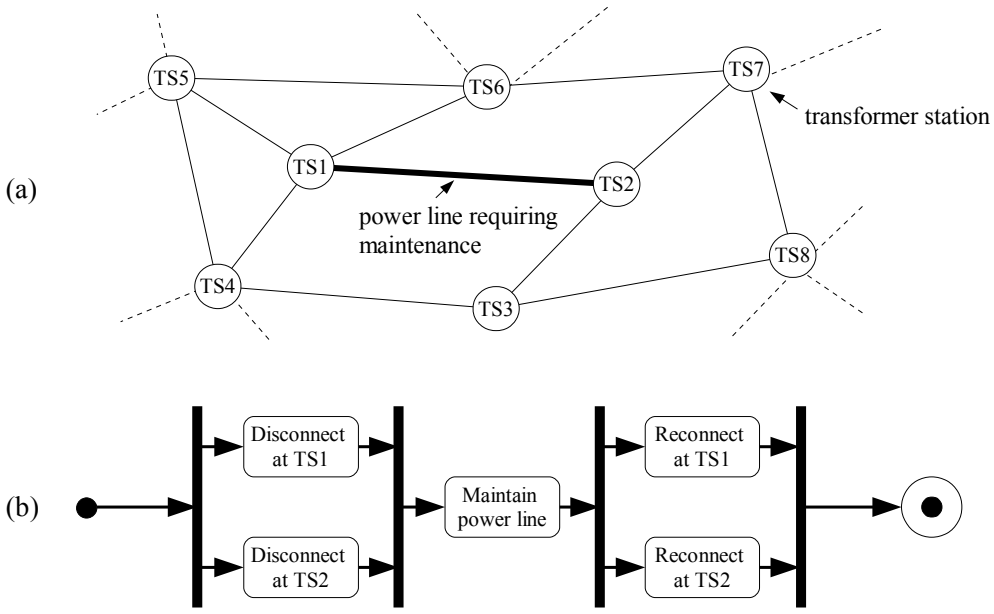

Figure 1: Part of a network (a) and interdependent tasks for maintaining power line TS1-TS2 (b).

are to be performed at different locations minimal downtimes can only be achieved if different workers are assigned to the different tasks and if all workers synchronize their activities.

The corresponding optimization problem that is addressed in this paper can be described as follows. Given a set of maintenance jobs that have to be performed by a given set of workers the problem is to find an assignment of tasks to workers and a schedule for performing the tasks such that downtimes and travel effort are minimized. The paper's contributions are to introduce this optimization problem, to provide a corresponding mathematical formulation, and to present various methods for its solution. The methods include local search techniques and a matheuristic that combines a Large Neighborhood Search (LNS) meta-heuristic for the routing of workers with mathematical programming techniques for the scheduling of tasks. We test the methods on network data of a large electricity provider operating in central Germany.

The paper is organized as follows. In Section 2, we review the related literature. The problem of routing and scheduling workers for maintenance in electricity networks is then formally described and mathematically modeled in Section 3. Section 4 presents the heuristic solution methods that we have developed for solving the optimization problem. A comprehensive computational study is provided in Section 5. Section 6 concludes the paper.

2. Literature

Maintenance operations in electricity networks must be carefully planned to minimize the impact on the users of the network. Planning and scheduling of preventive maintenance activities is often viewed from a tactical perspective and maintenance activities are scheduled in such a way that the cost and impact of preventive maintenance are as small as possible while minimizing the likelihood that failures occur due to long periods without maintenance. Surveys of such models are given by Percy (2008) and Budai et al. (2008). Papers addressing the interrelations of maintenance planning and product warranty offered to customers are surveyed in Shafiee & Chukova (2013). Contrasting such long-term maintenance issues, we focus on the operational decision level, i.e. planning of the actual execution of maintenance activities. We assume that tactical maintenance planning has already been conducted and, thus, it is known which maintenance activities need to be performed within the planning horizon. Unlike Budai et al. (2006) who present a preventive

maintenance scheduling problem for railway maintenance in which the goal is to group together different maintenance activities on the same railway link, we assume that this grouping of different tasks into jobs is already given and that the main operational goal is to assign workers to the different maintenance tasks in such a way that all jobs are performed at minimal travel costs and job downtimes. Peng et al. (2011) study a railway track maintenance problem in which the objective is to minimize travel costs of maintenance teams required to perform geographically dispersed maintenance tasks. The duration of each maintenance job depends on the maintenance team assigned to the job and the teams cannot be split. The problem studied in our paper differs from their problem as we seek to minimize travel time and the time required for each maintenance job. Furthermore, workers are assigned to individual tasks in our problem which allows them to switch back and forth between different maintenance jobs.

If each maintenance job consists of exactly one task, the problem studied in this paper reduces to a vehicle routing problem. Comprehensive surveys on the vehicle routing problem and some of its most important variants are given in Toth & Vigo (2002). There are several generalizations of the vehicle routing problem in which each transportation request consists of more than one task. In the pickup and delivery problem (Parragh et al., 2008a;b) each transportation request consists of exactly one pickup task and one delivery task. In the general pickup and delivery problem (Savelsbergh & Sol, 1995) and the general vehicle routing problem (Goel & Gruhn, 2008) transportation requests may include multiple pickup and delivery tasks. These problems have in common that all tasks belonging to one transportation request must be executed by the same vehicle. In the problem considered in our paper, however, tasks belonging to the same job can be executed by different workers in parallel to reduce the job downtime. If tasks of the same job can be assigned to different workers and if precedence relations exist for these tasks, the routes of the workers must be synchronized. Routing problems with such synchronization requirements are surveyed by Drexl (2012). However, a typical feature of these problems is that all tasks of a job occur at the same location, see e.g. Kim et al. (2010). Hence, travel times between geographically dispersed tasks and their impact on job durations are out of scope of these studies.

Scheduling multi-task jobs together with the objective of minimizing job durations is well known in the field of resource-constrained project scheduling (Kolisch, 1996). Considering main-

tenance jobs as projects and workers as resources, the problem investigated in our paper can be seen as a resource-constrained multi-project scheduling problem as investigated for example in Gonçalves et al. (2008). However, geographically dispersed tasks and routing decisions for workers are not supported in this problem class. Such a feature is covered by the generalized multi-project scheduling problem proposed in Krüger & Scholl (2009), where the time for moving resources between tasks of a job is captured in the planning by sequence-dependent transfer times. In this approach, due date related objectives are pursued and, thus, heuristics are proposed that aim at starting jobs as early as possible. The objective in our problem is to minimize the downtime of network assets, which calls for solution methods that can postpone start times of tasks if this leads to shorter job durations. We will show that this is one of the main challenges encountered in solving the optimization problem considered here.

Related studies are also found in the research field of workforce scheduling problems. These problems mostly address the fulfillment of staff requirements by assigning workers to the work shifts of a company (see the survey of Van den Bergh et al., 2013) but operational problems of scheduling work jobs that are to be processed by a set of workers also exist. The objective of the latter studies is typically to minimize the number of required workers and their travel effort, or to maximize the number of tasks that can be processed within given time windows (Li et al., 2005, Dohn et al., 2009, Cowling et al., 2006). The approaches often support different skills of workers which must be considered, for example, in maintenance operations for telecommunication companies studied by Cordeau et al. (2010). Multi-task jobs and the minimization of job durations is out of scope of these problems.

The combined problem of routing workers and scheduling tasks for a setting with multi-task jobs, dispersed task locations and the objective of minimizing traveling of workers and downtimes of jobs has not been considered so far. Especially the application of such a problem in the field of electricity network maintenance has merely been introduced in our previous study (Goel et al., 2010), but models and solution methods for such problems are still lacking.

3. The Network Maintenance Problem

We are given w workers who have to perform a set of maintenance jobs $\mathcal{J}$ within one work shift. For each job $j \in \mathcal{J}$ let $\mathcal{T}_j$ denote the set of tasks belonging to the job and let $\mathcal{T} = \bigcup_{j \in \mathcal{J}} \mathcal{T}_j$ denote the set of all tasks. For the tasks belonging to job $j \in \mathcal{J}$, let $\mathcal{P}_j \subset \mathcal{T}_j \times \mathcal{T}_j$ denote a set of precedence constraints. Each precedence constraint $(\tau, \tau') \in \mathcal{P}_j$ requires that task τ is completed before task τ' is started. The set of all precedence constraints is denoted by $\mathcal{P} = \bigcup_{j \in \mathcal{J}} \mathcal{P}_j$.

All workers are homogeneous in a sense that each worker is able to perform each task and that travel times, travel cost, and service times are identical for all workers. They are initially located at a depot denoted by 0. With $\mathcal{N} = \{0\} \cup \mathcal{T}$ we denote the set of locations that are of relevance for the problem and with $\mathcal{A} = \mathcal{N} \times \mathcal{N} \setminus \{(n, n) \mid n \in \mathcal{N}\}$ we denote the set of arcs connecting pairs of locations. For each arc $(n, m) \in \mathcal{A}$, let $c_{n,m}$ and $d_{n,m}$ denote the nonnegative cost and

Table 1: Notation used for modeling the planning problem.

Sets:	
$\mathcal{J}$	set of all maintenance jobs,
$\mathcal{T}_j$	set of tasks belonging to job $j \in \mathcal{J}$,
$\mathcal{T}$	set of all tasks, $\mathcal{T} = \bigcup_{j \in \mathcal{J}} \mathcal{T}_j$,
$\mathcal{P}_j$	set of precedence constraints among the tasks of job j, $\mathcal{P}_j \subset \mathcal{T}_j \times \mathcal{T}_j$,
$\mathcal{P}$	set of all precedence constraints, $\mathcal{P} = \bigcup_{j \in \mathcal{J}} \mathcal{P}_j$,
$\mathcal{N}$	set of nodes, $\mathcal{N} = \{0\} \cup \mathcal{T}$,
$\mathcal{A}$	set of arcs,
Parameters:	
w	number of available workers,
s_n	service time required by a worker to execute the task at node $n \in \mathcal{N}$,
$d_{n,m}$	duration for traveling along arc $(n, m) \in \mathcal{A}$,
$c_{n,m}$	cost for traveling along arc $(n, m) \in \mathcal{A}$,
c_j	cost per minute of downtime of job j,
λ	objective function weight,
T	planning horizon, i.e., the duration of the work shift,
Decision variables:	
$x_{n,m}$	binary, 1 if any worker travels along arc $(n, m) \in \mathcal{A}$,
t_τ	start time of task τ, $t_\tau \geq 0$,
t_j^-	earliest start time among the tasks belonging to job $j \in \mathcal{J}$, $t_j^- \geq 0$,
t_j^+	latest completion time among the tasks belonging to job j, $t_j^+ \geq 0$.

duration for having a worker travel from n to m. We assume that travel times fulfill the triangle inequality. Fixed cost for each worker leaving the depot can be considered by including them into the arc costs $c_{0,n}$ for all $n \in \mathcal{T}$. At each node $n \in \mathcal{T}$ a service operation of duration $s_n > 0$ must be conducted. For the ease of notation we assume that $s_0 = 0$, i.e. no work must be conducted at the depot. All workers must have returned to the depot by the end of the planning horizon which is denoted by T.

The main decisions to be made are the assignment of tasks to workers and the timing of the tasks. To model the corresponding decisions, we denote with $x_{n,m}$ a binary variable that takes value 1 if a worker travels on arc $(n, m) \in \mathcal{A}$ and with t_τ a continuous variable representing the start time of task $\tau \in \mathcal{T}$. For the ease of notation we assume that we can set $t_0 = 0$, i.e. we assume that all workers can leave the depot at the beginning of the planning horizon. From the start times of the tasks, the start time of job j is determined by $\min_{\tau \in \mathcal{T}_j}\{t_\tau\}$ and the completion time of the job is determined by $\max_{\tau \in \mathcal{T}_j}\{t_\tau + s_\tau\}$. We introduce for each job $j \in \mathcal{J}$ two continuous variables t_j^- and t_j^+ that indicate these start and completion times. The downtime of the network asset that is shut down for performing job j can then be determined by $t_j^+ - t_j^-$. For each job $j \in \mathcal{J}$ the cost for each minute of downtime is denoted by c_j. The objective of the optimization problem is to minimize the weighted sum of downtime costs and travel costs. A parameter λ with $0 \leq \lambda \leq 1$ is used to balance downtime and travel costs. The introduced notation is summarized in Table 1.

The joint optimization problem of determining work plans for the maintenance workers and of scheduling the maintenance tasks is formulated as follows.

$$\text{minimize } \lambda \sum_{j \in \mathcal{J}} c_j (t_j^+ - t_j^-) + (1 - \lambda) \sum_{(n,m) \in \mathcal{A}} c_{n,m} x_{n,m} \tag{1}$$

subject to

$$\sum_{(0,m) \in \mathcal{A}} x_{0,m} \leq w \tag{2}$$

$$\sum_{(n,m) \in \mathcal{A}} x_{n,m} = 1 \qquad \text{for all } n \in \mathcal{T} \tag{3}$$

$$\sum_{(n,m) \in \mathcal{A}} x_{n,m} = \sum_{(m,n) \in \mathcal{A}} x_{m,n} \qquad \text{for all } n \in \mathcal{N} \tag{4}$$

$$t_n + s_n + d_{n,m} \leq t_m + M(1 - x_{n,m}) \qquad \text{for all } n \in \mathcal{N}, m \in \mathcal{T} \setminus \{n\} \tag{5}$$

$$t_n + s_n + d_{n,0} \leq T \qquad \text{for all } n \in \mathcal{T} \tag{6}$$

$$t_j^- \leq t_\tau \qquad \text{for all } j \in \mathcal{J}, \tau \in \mathcal{T}_j \tag{7}$$

$$t_j^+ \geq t_\tau + s_\tau \qquad \text{for all } j \in \mathcal{J}, \tau \in \mathcal{T}_j \tag{8}$$

$$t_\tau + s_\tau \leq t_{\tau'} \qquad \text{for all } (\tau, \tau') \in \mathcal{P} \tag{9}$$

$$x_{n,m} \in \{0, 1\} \qquad \text{for all } (n, m) \in \mathcal{A} \tag{10}$$

$$t_j^-, t_j^+ \in [0, T] \qquad \text{for all } j \in \mathcal{J} \tag{11}$$

$$t_\tau \in [d_{0,\tau}, T] \qquad \text{for all } \tau \in \mathcal{T} \tag{12}$$

$$t_0 = 0 \tag{13}$$

The objective (1) is to minimize the weighted sum of total downtime cost of the assets and total travel cost of all workers. Constraint (2) ensures that no more than w workers leave the depot. Constraint (3) demands that each maintenance task is performed exactly once. Constraint (4) balances the flow of workers at the nodes of the network. Constraint (5) demands that the time between visiting two nodes subsequently by the same worker is sufficient to allow the worker performing the task at the first node and to travel to the geographical location of the second node. In this constraint, M refers to a sufficiently large positive value, e.g. $M = T$. Constraint (6) guarantees that all workers return to the depot no later than time T. Constraints (7) and (8) require that no task belonging to a job begins before the start time of the job and that no task is completed after the completion time of the job. Note that in any optimal solution of a problem with $\lambda > 0$ and a job $j \in \mathcal{J}$ with $c_j > 0$, we have $t_j^- = \min_{\tau \in \mathcal{T}_j}\{t_\tau\}$ and $t_j^+ = \max_{\tau \in \mathcal{T}_j}\{t_\tau + s_\tau\}$. Precedence constraints are respected through (9). Domains of the decision variables are defined in (10) to (13).

Considering the special case of the problem with $w = 1$, $|\mathcal{T}_j| = 1$ for all $j \in \mathcal{J}$, $s_n = 0$ for all $n \in \mathcal{N}$, $T = \infty$, and $\lambda = 0$ the problem reduces to the traveling salesman problem which is known to be $\mathcal{NP}$-hard (Karp, 1972). Therefore, the problem studied in this paper is also $\mathcal{NP}$-hard.

Example: We consider an illustrative example with three workers, three jobs $\mathcal{J} = \{1, 2, 3\}$, and a planning horizon of $T = 120$ time units. Each job consists of five tasks and has the same structure as the job illustrated in Fig. 1. The service times for disconnecting or reconnecting a power line at a transformer station are 10 minutes and the actual maintenance operation requires

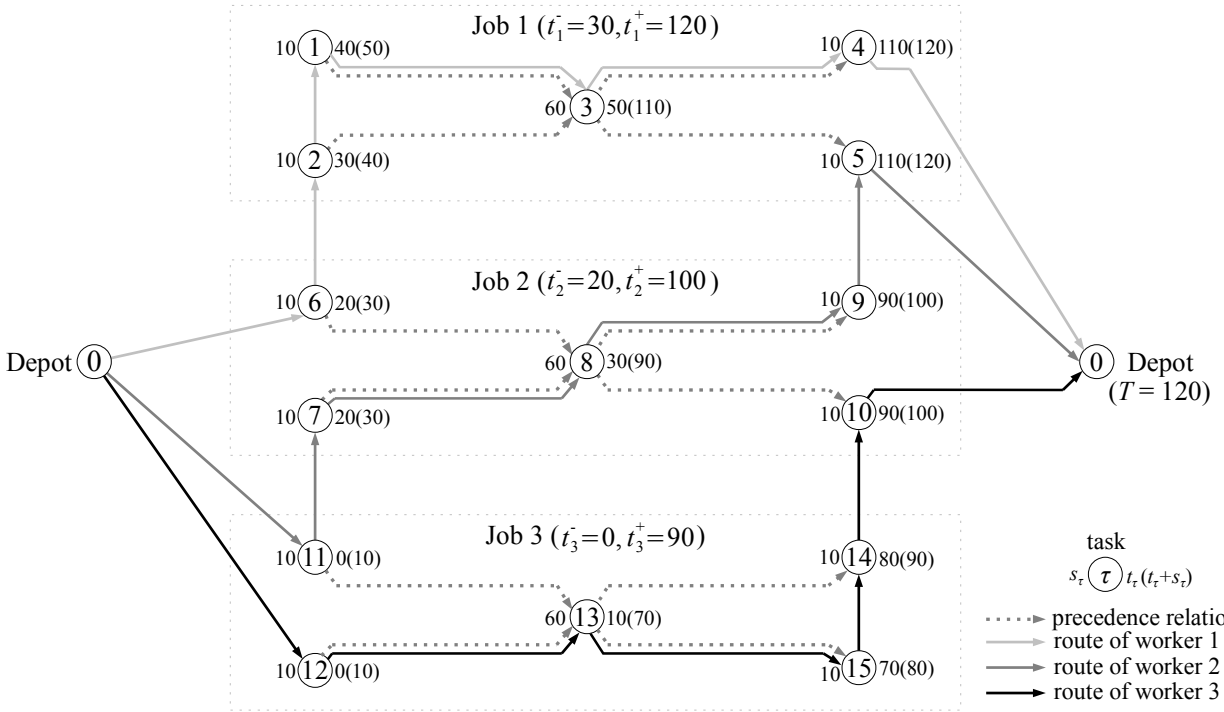

Figure 2: Example.

60 minutes. For reasons of simplicity, travel times and cost are ignored here, i.e. $d_{n,m} = c_{n,m} = 0$ for all $(n, m) \in \mathcal{A}$, and we focus on minimizing total downtime ($\lambda = 1$ and $c_j = 1$ for all $j \in \mathcal{J}$).

Figure 2 illustrates an optimal solution to the problem. For each task $\tau \in \mathcal{T}$ the service time s_τ is indicated by the label to the left of the node and the start time t_τ and the corresponding completion time $t_\tau + s_\tau$ are indicated by the label to the right of the node. The routes of workers are 0-6-2-1-3-4-0, 0-11-7-8-9-5-0, and 0-12-13-15-14-10-0. The resulting downtimes of jobs 1, 2, and 3 are 90, 80, and 90 time units, respectively. As tasks 1 and 2 as well as tasks 14 and 15 are in the route of the same worker, the downtimes of jobs 1 and 3 are 10 minutes higher than the theoretical lower bound on the total downtime of these jobs which could be achieved if more workers were available or if the planning horizon was larger. It must also be noted that the optimal solution cannot be obtained by starting all tasks at their earliest possible start times. In this example, tasks 6 and 7 could be started at times 0 and 10, respectively. However, the start times of these tasks in the optimal solution are at time 20 because task 10 cannot be performed before time 90 and smaller start times of tasks 6 and 7 would lead to an increased downtime of job 2.

4. Solution Approach

As the problem studied in this paper is $\mathcal{NP}$-hard it is in general not practical to solve the problem using a MIP-solver. In this section we present a solution approach based on decomposition where heuristics are used for determining the values of the integer variables of the original problem and linear programming (LP) is used for determining the values of the continuous variables of the original problem. Such a hybridized method is known as matheuristic, cf. Maniezzo et al. (2009). It combines advantages of meta-heuristic search (i.e., quickly exploring different areas of a solution space) with those of mathematical programming (i.e., solving subproblems to optimality).

In the above example we observed that the best start time of a task depends on the start times of other tasks. Thus, finding optimal start times for all tasks is not trivial even if the route of each worker is known. Let us assume that the values of $x_{n,m}$ are fixed for all $(n,m) \in \mathcal{A}$ and that the values satisfy constraints (2) to (4) and (10). In the following we will refer to such a variable vector $(x_{n,m})_{(n,m)\in\mathcal{A}}$ as a *work plan*. The problem of determining optimal start times for a given work plan is

$$\text{minimize} \sum_{j\in\mathcal{J}} c_j(t_j^+ - t_j^-) \tag{14}$$

subject to

$$t_n + s_n + d_{n,m} \leq t_m \qquad \text{for all } n \in \mathcal{N}, m \in \mathcal{T}\setminus\{n\} : x_{n,m} = 1 \tag{15}$$

$$t_n + s_n + d_{n,0} \leq T \qquad \text{for all } n \in \mathcal{T} \tag{16}$$

$$t_\tau + s_\tau \leq t_{\tau'} \qquad \text{for all } (\tau,\tau') \in \mathcal{P} \tag{17}$$

$$t_\tau \in [d_{0,\tau}, T] \qquad \text{for all } \tau \in \mathcal{T} \tag{18}$$

$$t_j^- \leq t_\tau \qquad \text{for all } j \in \mathcal{J}, \tau \in \mathcal{T}_j \tag{19}$$

$$t_j^+ \geq t_\tau + s_\tau \qquad \text{for all } j \in \mathcal{J}, \tau \in \mathcal{T}_j \tag{20}$$

$$t_j^-, t_j^+ \in [0, T] \qquad \text{for all } j \in \mathcal{J} \tag{21}$$

As this problem only contains continuous variables, linear constraints, and a linear objective function it can be solved using any linear programming solver. For any set of start times $(t_\tau)_{\tau\in\mathcal{T}}$ satisfying constraints (15) to (18), we know that $((t_\tau)_{\tau\in\mathcal{T}}, (t_j^-)_{j\in\mathcal{J}}, (t_j^+)_{j\in\mathcal{J}})$ with $t_j^- =$

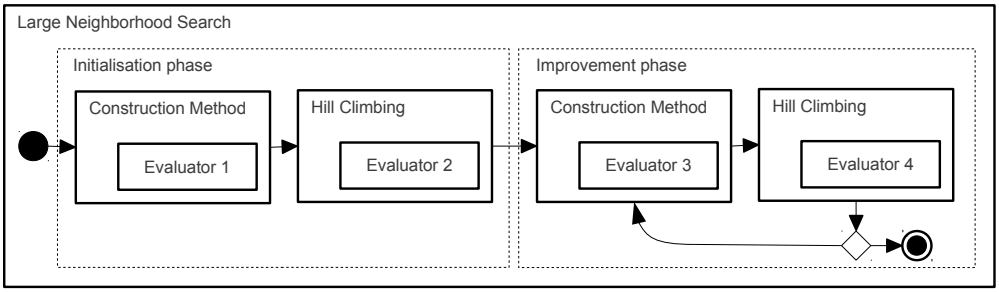

Figure 3: Overview of the solution approach.

$\min_{\tau \in \mathcal{T}_j}\{t_\tau\}$ and $t_j^+ = \max_{\tau \in \mathcal{T}_j}\{t_\tau + s_\tau\}$ for all $j \in \mathcal{J}$ is a feasible solution to the overall optimization problem. Furthermore, this assignment of start and completion times of jobs results in the smallest objective function value. In the remainder we will refer to a set of start times $(t_\tau)_{\tau \in \mathcal{T}}$ as a *schedule*. We say that a schedule is *feasible* if and only if it satisfies constraints (15) to (18), and we say that a schedule is *optimal* if and only if it is feasible and together with $t_j^- = \min_{\tau \in \mathcal{T}_j}\{t_\tau\}$ and $t_j^+ = \max_{\tau \in \mathcal{T}_j}\{t_\tau + s_\tau\}$ for all $j \in \mathcal{J}$ is an optimal solution to the problem defined by (14) to (21).

A work plan $(x_{n,m})_{(n,m)\in\mathcal{A}}$ satisfying constraints (2) to (4) and (10) is *feasible* if and only if a feasible schedule exists for the work plan. The work plan is *optimal* if a feasible schedule exists for the work plan and the costs defined by (1) are minimized when using the optimal schedule.

The network maintenance problem can thus be heuristically solved by iteratively generating and modifying work plans, determining an optimal schedule for each work plan using an LP solver, and by memorizing the best work plan found within the solution process. Figure 3 provides an overview of the solution approach proposed in this paper. The method begins with an initialization phase in which a first feasible solution is generated from scratch by using the construction method described in Section 4.1. After construction, a hill climbing method (Section 4.2) is used to obtain a locally optimal solution. In the improvement phase, the solution is partially destroyed and, then, rebuilt using the same methods as in the initialization phase. How this is done is described in Section 4.3. Within all building blocks of the resulting Large Neighborhood Search method, feasibility of (partial) work plans and their associated schedules are determined using an evaluator, e.g. a method using an LP solver for the scheduling problem described above. Different evaluators are described in Section 4.4.

4.1. Construction Method

In order to generate a work plan from scratch let us consider the relaxed problem obtained by replacing constraint (3) with

$$\sum_{(n,m)\in\mathcal{A}} x_{n,m} \leq 1 \qquad \text{for all } n \in \mathcal{T} \tag{22}$$

and let us say that $(x_{n,m})_{(n,m)\in\mathcal{A}}$ satisfying constraints (2), (4), (10) and (22) is a *partial work plan*. Feasibility of a partial work plan is defined analogously to feasibility of a work plan. However, in order to determine a schedule for a partial work plan, we have to replace in the scheduling problem defined by (14) to (21) the set of nodes $\mathcal{N}$ with the set of nodes containing only those tasks which are already assigned to a worker.

The algorithm illustrated in Fig. 4 attempts constructing a feasible work plan x by iteratively inserting tasks into a partial work plan until condition (3) is satisfied. For the ease of notation we include the element $x_{0,0}$ in the vector x. Within the course of the algorithm $x_{0,0}$ indicates the number of workers who are not assigned to any task. The construction method is invoked with an initially empty work plan which is obtained by setting $x_{0,0} = w$ and $x_{n,m} = 0$ for all arcs $(n, m) \in \mathcal{A}$. Furthermore, one of the evaluators described in Section 4.4 is specified to be used to assess the quality of (partial) work plans.

construct(x, `evaluator`)

for all $j \in \mathcal{J}$ **do**
  while $\{\tau \in \mathcal{T}_j \mid \sum_{(n,\tau)\in\mathcal{A}} x_{n,\tau} = 0\} \neq \emptyset$ **do** ▷ Any unserved tasks remaining?
    select $\tau \in \mathcal{T}_j$ with $\sum_{(n,\tau)\in\mathcal{A}} x_{n,\tau} = 0$ and $\sum_{(n,\tau')\in\mathcal{A}} x_{n,\tau'} = 1$ for all $(\tau', \tau) \in \mathcal{P}_j$;
    set `best` $:= \infty$;
    for all $(n, m) \in \mathcal{A} \cup \{(0, 0)\}$ with $x_{n,m} > 0$ **do**
      set $x^+ := x$, $x^+_{n,m} := x^+_{n,m} - 1$, $x^+_{n,\tau} := 1$, and $x^+_{\tau,m} := 1$; ▷ insert τ between n and m
      if $f(x^+,$ `evaluator`$) <$ `best` **then**
        `best` $:= f(x^+,$ `evaluator`$)$ and $x^* := x^+$;
    if `best` $= \infty$ **then** stop; ▷ no feasible insertion possible
    set $x := x^*$;
return x;

Figure 4: Construction method.

The algorithm iteratively selects the first unscheduled job $j \in \mathcal{J}$ from a list, removes it from the list, and conducts the following steps. As long as there are tasks in $\mathcal{T}_j$ which are not yet assigned to a worker, the algorithm selects one such task for which all predecessor tasks are already contained in the partial plan. The choice of the next task is realized by selecting the first unscheduled task in $\mathcal{T}_j$ from an ordered list which is generated in such a way that all predecessor tasks have an earlier position in the list than their successors. For all arcs $(n, m) \in \mathcal{A} \cup \{(0, 0)\}$ with $x_{n,m} > 0$ the algorithm inserts the task between n and m and determines whether the resulting work plan x^+ is feasible. The quality of all feasible work plans is determined using an evaluation function $f(\cdot, \cdot)$. Different evaluation functions that can be used for this purpose are described in Section 4.4. Each work plan showing an improvement is accepted as new work plan. If the task cannot be inserted the algorithm terminates prematurely. Otherwise, it continues with inserting the next task of the job, or, if no such task exists, with the next job. If all tasks of all jobs are inserted the algorithm terminates and returns a feasible work plan.

Note that the evaluation of a partial work plan is not trivial because the exact downtimes of jobs can only be determined if all their tasks are included in a plan. When trying to generate a feasible work plan from scratch by subsequently inserting tasks, we must pay special attention to the precedence constraints associated to tasks of the same job. If a task $\tau \in \mathcal{T}_j$ is executed at time t_τ, all other tasks $\tau' \in \mathcal{T}_j$ with $(\tau, \tau') \in \mathcal{P}_j$ must start at some time in $[t_\tau + s_\tau, T]$. If t_τ is very large, it is unlikely that all subsequent tasks can be executed within the planning horizon. In Section 4.4 we therefore not only present evaluators focusing on the objective function value, but also an evaluator focusing on the likelihood that a feasible work plan can be found.

4.2. Hill Climbing

Depending on the evaluation function and the sequence in which jobs are selected for insertion, the quality of work plans obtained by the construction method can vary. Given a feasible work plan, we can improve its quality using the hill climbing method shown in Figure 5. The method is invoked with a feasible work plan and an evaluator which can be different to the evaluator used during construction. Note that like for the construction method we include $x_{0,0}$ within the vector x to indicate the number of workers who are not assigned to any task.

hillclimbing(x, `evaluator`)

set $\mathcal{T}^* := \mathcal{T}$;
while $\mathcal{T}^* \neq \emptyset$ **do**
  select $\tau \in \mathcal{T}^*$ and set $\mathcal{T}^* := \mathcal{T}^* \setminus \{\tau\}$;
  set `best` $:= \infty$ and $x' := x$;
  let n and m denote nodes with $x'_{n,\tau} = x'_{\tau,m} = 1$;
  set $x'_{n,m} := x'_{n,m} + 1$, $x'_{n,\tau} := 0$, and $x'_{\tau,m} := 0$; ▷ remove τ from work plan
  for all $(n, m) \in \mathcal{A} \cup \{(0, 0)\}$ with $x'_{n,m} > 0$ **do**
    set $x^+ := x'$, $x^+_{n,m} := x^+_{n,m} - 1$, $x^+_{n,\tau} := 1$, and $x^+_{\tau,m} := 1$; ▷ insert τ between n and m
    if $f(x^+,$ `evaluator`$) <$ `best` **then**
      set `best` $:= f(x^+,$ `evaluator`$)$ and $x^* = x^+$;
  if $x^* \neq x$ **do**
    set $x := x^*$ and $\mathcal{T}^* := \mathcal{T}$; ▷ new best solution restarts search
return x;

Figure 5: Hill climbing method.

The hill climbing method randomly selects a task and removes it from the work plan. Then it searches for the best way to re-insert the task with respect to the evaluation function. If the best re-insertion leads to a modified work plan, this modified work plan must be at least as good as the previous one and replaces the latter. The process is repeated by selecting the next task until all tasks have been removed and re-inserted without any improvement.

Within the hill climbing approach we know that any task which is removed from a feasible work plan can be feasibly re-inserted - if necessary at the same position as before. Therefore, we do not need to consider the likelihood that other tasks can be inserted and the evaluation function used during hill climbing can focus on the costs of the solution.

4.3. Large Neighborhood Search

After construction and hill climbing, a locally optimal solution is found which may still be of insufficient quality. To further improve solution quality we apply Large Neighborhood Search (LNS). LNS attempts escaping from local optima by using large neighborhood moves that involve numerous tasks, see Shaw (1997). The LNS algorithm for our problem is illustrated in Fig. 6. The algorithm starts by generating an initial solution from scratch using the construction method. Then, the solution quality is improved by applying hill climbing as described above. The resulting

solution is iteratively improved by randomly removing a preset number of jobs and re-inserting them using the construction method and, afterwards, by applying hill climbing. The algorithm terminates when the number of iterations reaches a given limit. In each phase of the algorithm evaluators with different focus on feasibility, solution quality, and computation speed are used to obtain an overall efficient and effective algorithm.

4.4. Feasibility Check and Evaluation of Work Plans

To determine whether a given work plan is feasible we can use an LP solver to solve the scheduling problem defined by (14) to (21). If a feasible schedule for the work plan exists the solver will find the optimal schedule. Otherwise, it will terminate indicating that no feasible schedule exists. The computational effort required by a general purpose linear programming solver is, however, unnecessarily high if the main goal is to determine whether a work plan is feasible or not.

We will now present a dynamic programming approach which can be used to quickly determine whether a feasible schedule exists for a given work plan. Let us consider the auxiliary graph containing nodes $\mathcal{N}$ and the arcs $\mathcal{A}' := \{(n,m) \in \mathcal{A} \mid m \neq 0, x_{n,m} = 1\} \cup \mathcal{P}$. The arcs in the auxiliary graph represent the precedence relationships corresponding to the work plan and the

LNS(`evaluator1`, `evaluator2`, `evaluator3`, `evaluator4`, k, i)

1: set $x_{0,0} := w$ and $x_{n,m} := 0$ for all arcs $(n,m) \in \mathcal{A}$;
2: set $x :=$ construct$(x,$ `evaluator1`$)$;
3: set $x :=$ hillclimbing$(x,$ `evaluator2`$)$;
4: determine optimal schedule for x and resulting objective function value;
5: **while** iteration counter $\leq i$ **do**
6: set $x^+ :=$ remove(x,k); ▷ Remove k jobs from work plan x
7: set $x^+ :=$ construct$(x^+,$ `evaluator3`$)$;
8: set $x^+ :=$ hillclimbing$(x^+,$ `evaluator4`$)$;
9: determine optimal schedule for x^+ and resulting objective function value;
10: **if** x^+ has better objective function value than x **then**
11: set $x := x^+$;
12: **return** x;

Figure 6: Large Neighborhood Search.

jobs. If the auxiliary graph contains a directed cycle the work plan is infeasible as precedence constraints are violated. Otherwise, we can determine the earliest start time for each node by iterating down a spanning tree starting from node 0. If the earliest start time at any node violates constraint (16) or (18), no feasible schedule exists for the work plan. If constraints (15) to (18) are satisfied for the earliest start time of each node, a feasible schedule is found.

check(x)

1: set $\mathcal{A}' := \{(n,m) \in \mathcal{A} \mid m \neq 0, x_{n,m} = 1\} \cup \mathcal{P}$;
2: set $t_0 := 0, \mathcal{N}^* := \{0\}$;
3: **while** $\mathcal{N}^* \neq \mathcal{N}$ **do**
4: **if** $\{m \in \mathcal{N} \setminus \mathcal{N}^* \mid n \in \mathcal{N}^*$ for all $(n,m) \in \mathcal{A}'\} = \emptyset$ **return** false;
5: select $m \in \mathcal{N} \setminus \mathcal{N}^*$ with $n \in \mathcal{N}^*$ for all $(n,m) \in \mathcal{A}'$;
6: let n denote the node with $x_{n,m} = 1$;
7: set $t_m := \max\{t_n + s_n + d_{n,m}, \max_{(n',m)\in\mathcal{P}}\{t_{n'} + s_{n'}\}\}$;
8: **if** $t_m + s_m + d_{m,0} > T$ **return** false;
9: set $\mathcal{N}^* := \mathcal{N}^* \cup \{m\}$;
10: **return** true;

Figure 7: Feasibility check.

The algorithm illustrated in Fig. 7 determines in $|\mathcal{T}|$ iterations whether the work plan represented by variables x is feasible or not. The algorithm begins by constructing the auxiliary graph and by setting the start time at the depot to zero and by initializing the set $\mathcal{N}^* = \{0\}$ of visited nodes. As long as there is a node in $\mathcal{N}$ which is not yet included in $\mathcal{N}^*$ the method iterates with the following steps. First, it checks whether there is a node in $\mathcal{N} \setminus \mathcal{N}^*$ for which all predecessor nodes in the auxiliary graph are already included in $\mathcal{N}^*$. If no such node exists, then each not yet included task must wait for another not yet included task to be completed. Thus, the auxiliary graph contains a directed cycle and the algorithm stops because the work plan is infeasible. Otherwise, the algorithm selects such a node and determines the start time of the corresponding task considering the start times of all predecessor nodes in the auxiliary graph. If it is impossible to return to the depot after completion of the task the algorithm stops because the work plan is infeasible. Otherwise, it includes the task in the set $\mathcal{N}^*$ and continues with the next iteration.

Figure 8 illustrates the auxiliary graph and a feasible schedule computed by the algorithm. In the first iteration node 6 is selected as this is one of the nodes where all incoming arcs originate

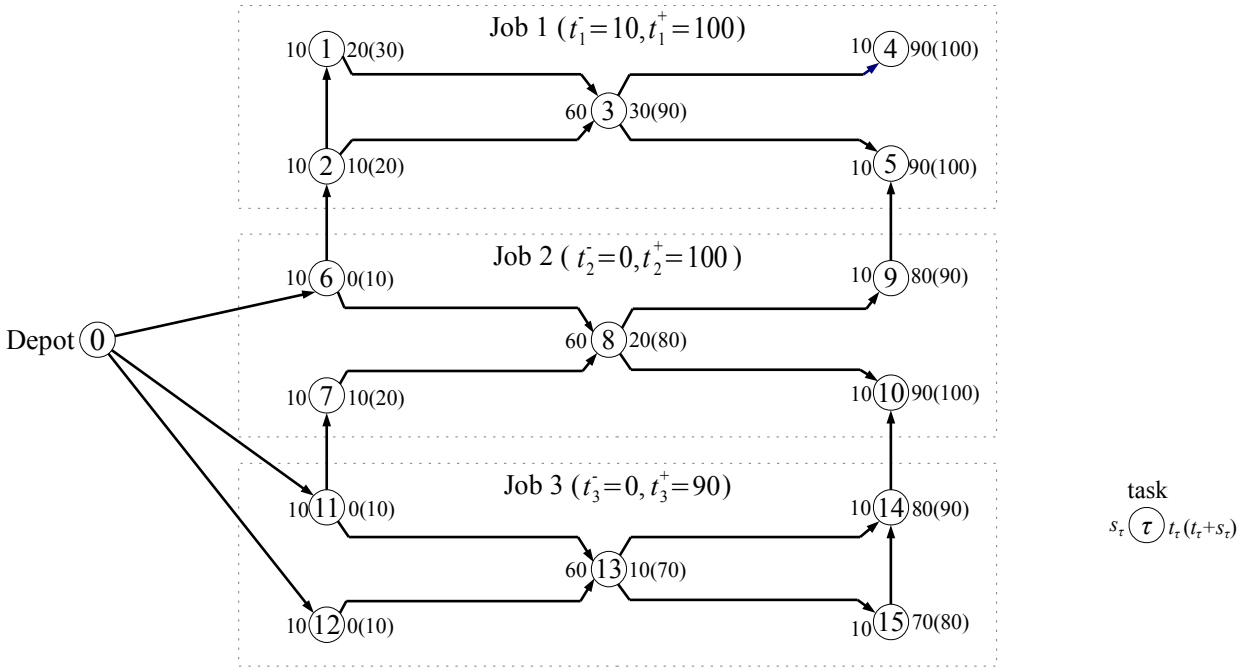

Figure 8: Auxiliary graph.

at a node from $\mathcal{N}^*$. The start time at the node is computed and the node is included in $\mathcal{N}^*$. In the next iterations, the nodes are selected in the order 2, 1, 3, 4, 11, 7, 8, 9, 5, 12, 13, 15, 14, and 10 and the earliest start time at each node can be uniquely determined considering start times of nodes in $\mathcal{N}^*$ exclusively. Obviously, the nodes can also be selected in a different order as long as $n \in \mathcal{N}^*$ for all $(n, m) \in \mathcal{A}'$ if node $m \in \mathcal{N} \setminus \mathcal{N}^*$ is selected. As we can see, the total downtime of the schedule obtained by this algorithm is 280 units and thus 20 units larger than in the optimal solution shown in Fig. 2.

Although the algorithm cannot be used to determine an optimal schedule, it is often much faster to use this algorithm to determine whether a work plan is feasible or not. In the construction method and during hill climbing we iteratively (re-)insert tasks to a work plan. For each task there may be many different insertion possibilities and many of them are infeasible. Using the feasibility check described above can thus lead to noticeable savings in computational effort. As each alternative insertion possibility has a different impact on the objective function value and the likelihood of determining a feasible work plan at all we propose several evaluators which can be used in the different phases of the algorithm. Each evaluator checks whether the (partial) work

plan is feasible and returns a value that can be used to compare alternative (partial) work plans. If the (partial) work plan is infeasible the evaluators return infinity.

Optimal Schedule (OS): For each (partial) work plan this evaluator solves the scheduling problem using an LP solver to check feasibility and to find an optimal schedule. The evaluation function returns the objective function value associated to the (partial) work plan and this optimal schedule.

Feasibility Check & Optimal Schedule (FCOS): This evaluator uses the check() function described above to determine whether a (partial) work plan is feasible. If the (partial) work plan is feasible, the earliest start times obtained within the feasibility check are used as an additional lower bound on the start times within the scheduling problem. An optimal schedule is generated by solving the scheduling problem using an LP solver. The evaluation function returns the objective function value associated to the (partial) work plan and this optimal schedule.

Bidirectional Feasibility Check & Optimal Schedule (BFCOS): This evaluator uses the check() function described above to determine whether a (partial) work plan is feasible. If the (partial) work plan is feasible, a reverse version of the feasibility check is used to obtain the latest possible start time of each scheduled tasks. The earliest and latest start times obtained are used as additional lower and upper bounds on the start times within the scheduling problem. An optimal schedule is generated by solving the scheduling problem using an LP solver. The evaluation function returns the objective function value associated to the (partial) work plan and this optimal schedule.

Feasible Schedule (FS): This evaluator uses the check() function described above to determine a feasible schedule if possible. The evaluation function returns the objective function value associated to the (partial) work plan and this feasible schedule.

Minimal Start Times (MST): For a candidate (partial) work plan this evaluator applies the check() function described above to validate feasibility and to determine the earliest possible start time of each scheduled task. The evaluation function returns the sum over these earliest start times. As this evaluator privileges small start times when inserting tasks, the likelihood that all subsequent tasks of the same job can be inserted is increased. However, travel costs are totally ignored by this evaluator.

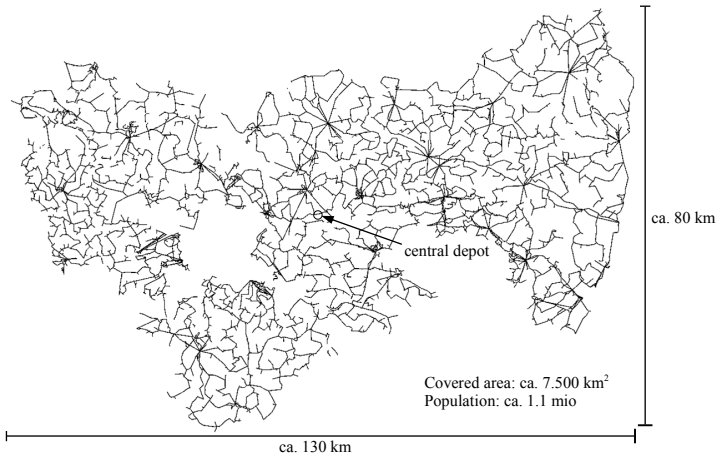

Figure 9: An electricity network in central Germany.

5. Computational Experiments

This section presents computational experiments assessing the usefulness of our approach and showing how to select evaluators within the different phases of the optimization process.

5.1. Test Instances

Our computational experiments are based on instances derived from real network data that has been provided to us by a company that operates a large electricity network in central Germany. The network, which is shown in Fig. 9, covers a rural area with an average population of 150 per km^2. The total length of power lines in this network is about 5200 km.

Using the network data, we have generated six instance sets $A1$ to $A3$ and $B1$ to $B3$ each containing 100 test instances. The instance sets differ in the number of jobs and tasks of the contained instances and in the regional area that is covered by the jobs, see Table 2. We randomly selected power lines from the given network and generated maintenance jobs for these power

Table 2: Instance sets.

set	nmbr. instances	nmbr. jobs $\|\mathcal{J}\|$	nmbr. tasks $\|\mathcal{T}\|$	covered area	set	nmbr. instances	nmbr. jobs $\|\mathcal{J}\|$	nmbr. tasks $\|\mathcal{T}\|$	covered area
$A1$	100	10	50	entire network	$B1$	100	10	50	subnetwork
$A2$	100	15	75	entire network	$B2$	100	15	75	subnetwork
$A3$	100	20	100	entire network	$B3$	100	20	100	subnetwork

lines. For instances in sets $A1$ to $A3$, these power lines are randomly selected within the area of the entire network. For instances in sets $B1$ to $B3$, power lines are selected from a randomly selected subnetwork that covers about one fourth of the overall network. Thus, travel distances in instance sets $B1$ to $B3$ are smaller than in $A1$ to $A3$. Each maintenance job j comprises five tasks $\mathcal{T}_j = \{\tau_1^j, \tau_2^j, \tau_3^j, \tau_4^j, \tau_5^j\}$ and has the structure illustrated in Fig. 1(b). Tasks τ_1^j and τ_2^j disconnect the power line from the network. They are located at the transformer stations linked to the power line. The actual maintenance is represented by task τ_3^j which we assume to be located in the middle of the power line. Tasks τ_4^j and τ_5^j reconnect the power line and are again located at the transformer stations. Precedences for all jobs are given by $\mathcal{P}_j = \{(\tau_1^j, \tau_3^j), (\tau_2^j, \tau_3^j), (\tau_3^j, \tau_4^j), (\tau_3^j, \tau_5^j)\}$. Processing times are set to $s_{\tau_1^j} = s_{\tau_2^j} = s_{\tau_4^j} = s_{\tau_5^j} = 10$ for tasks that dis- and reconnect a power line and to $s_{\tau_3^j} = 60$ for the maintenance task. The durations $d_{n,m}$ for traversing the network arcs $(n, m) \in \mathcal{A}$ are derived from the Euclidean distances with a travel speed of 60 km/h. Travel cost are set equal to travel times, i.e. $c_{n,m} = d_{n,m}$. The planning horizon is a 10-hour work shift, i.e. $T = 600$ minutes. All workers begin and end their shift at a depot which is located in the center of the network. Unless otherwise stated we set $\lambda = 0.5$ and $c_j = 1$ for all $j \in \mathcal{J}$ in order to consider both downtime of jobs and travel time of workers in the solution process. Alternative objective weights are later investigated in a sensitivity analysis.

The test instances are provided online at `http://prodlog.wiwi.uni-halle.de/electricity-maintenance`. For the experiments, the solution methods have been implemented in C++. As MIP and LP solver, we use CPLEX 12.4, see IBM (2012). The tests are performed on an Intel Core 2 Duo with 3.0 GHz.

5.2. Results for the Construction Method

The performance of the construction method presented in Section 4.1 depends on the evaluator which is used to compare partial work plans. If the evaluator focuses on the objective function value, some tasks of a job may be inserted at a very late time, reducing the likelihood that other tasks of the same job can be feasibly inserted due to precedence constraints. On the other hand, if the evaluator focuses on increasing the likelihood that all successor tasks of the same job can be feasibly inserted, the total travel time may be unnecessarily high. Thus, a lot of time may be

Table 3: Number of feasible initial solutions.

	set $A1$						set $A2$						set $A3$					
nmbr. workers $w \rightarrow$ *constructor*$\downarrow$	3	4	5	6	7	8	3	4	5	6	7	8	3	4	5	6	7	8
OS	57	66	66	66	66	66	0	11	32	33	33	33	0	0	0	18	22	22
FS	53	59	59	59	59	59	0	14	28	29	29	29	0	0	0	11	12	12
MST	0	73	**100**	**100**	**100**	**100**	0	0	6	69	97	99	0	0	0	0	17	71
OS^*	**91**	**100**	**100**	**100**	**100**	**100**	0	**33**	97	**100**	**100**	**100**	0	0	0	69	98	**100**
FS^*	88	99	**100**	**100**	**100**	**100**	0	**33**	**99**	**100**	**100**	**100**	0	0	**1**	**81**	**100**	**100**
	set $B1$						set $B2$						set $B3$					
nmbr. workers $w \rightarrow$ *constructor*$\downarrow$	3	4	5	6	7	8	3	4	5	6	7	8	3	4	5	6	7	8
OS	31	36	36	36	36	36	0	12	15	16	16	16	0	0	7	8	9	9
FS	37	40	40	40	40	40	0	11	17	17	17	17	0	0	4	6	6	6
MST	42	**100**	**100**	**100**	**100**	**100**	0	1	82	**100**	**100**	**100**	0	0	0	25	95	99
OS^*	**100**	**100**	**100**	**100**	**100**	**100**	0	84	**100**	**100**	**100**	**100**	0	0	33	97	**100**	**100**
FS^*	**100**	**100**	**100**	**100**	**100**	**100**	0	**91**	**100**	**100**	**100**	**100**	0	0	**49**	**99**	**100**	**100**

dedicated to traveling and less time remains for actually performing the tasks. Consequently, the likelihood that all jobs can be feasibly inserted may suffer.

In a first experiment we investigate the performance of the construction method with respect to the different evaluators outlined in Section 4.4. Table 3 shows the results of our experiment for the six instance sets and with a different number of workers available. The table shows the number of feasible solutions obtained (out of 100 instances) and indicates the best results in bold face. As evaluators OS, $FCOS$, and $BFCOS$ return the same value we only report results for OS.

For most instance sets evaluators OS and FS appear to perform better in finding feasible solutions than evaluator MST if the number of workers is small. With a small number of workers it appears to pay off using these evaluators as evaluator MST may result in unnecessarily large travel times occupying the available work time. On the other hand, evaluators OS and FS do not benefit much from the availability of additional workers, so that for larger number of workers MST can find more feasible schedules.

Table 3 also reports results of an additional variant of the construction method which combines the benefits of different evaluators. The construction method presented in Section 4.1 stops if any task cannot be feasibly inserted. As already mentioned above, the likelihood that all tasks of a job can be inserted depends on the evaluator used. The modified construction method conducts the

Table 4: Average costs of initial solutions.

constructor	set $A1\|w=5$	set $A2\|w=6$	set $A3\|w=8$	set $B1\|w=3$	set $B2\|w=5$	set $B3\|w=7$
OS^*	**870**	1341	1815	802	1240	1681
FS^*	880	**1327**	**1787**	**799**	**1214**	**1651**

same steps as the original method. However, if a task τ belonging to a job j cannot be inserted the modified method does not terminate directly. Instead it removes all tasks in $\mathcal{T}_j$ which may have been inserted before. Then it tries to insert all tasks belonging to j using evaluator MST. If successful, the method continues like the original method. Otherwise, it also terminates. In the table we denote with OS^* and FS^* the results of the modified construction method using evaluators OS and FS as primary evaluator. It can be seen that the modified construction method which uses evaluator MST if the primary evaluator fails, clearly outperforms the original method. It appears that FS^* performs better than OS^*, indicating that an exact evaluation of downtimes is not necessarily beneficial with respect to feasibility.

To further compare the performance of the different evaluators we assess the quality of the feasible solutions. For this comparison and all subsequent experiments we fix the number of workers available for each instance set to the smallest number for which both OS^* and FS^* succeed in finding feasible solutions for all 100 instances. Table 4 shows the average cost of initial solutions obtained for OS^* and FS^*. We see that neither evaluator clearly outperforms the other and that average costs differ by at most 2% with a slight advantage for evaluator FS^*.

5.3. Results for the Hill Climbing Method

As for the construction method, we need to select an evaluator for the hill climbing method. In a second experiment we investigate which combination of evaluators performs best for construction and subsequent hill climbing. The main difference in the choice of the evaluator used during hill climbing is that we already know that each task which is removed from a feasible work plan can be inserted again. Therefore, feasibility is not a concern during hill climbing and the evaluator used during hill climbing can focus fully on cost. As evaluator MST ignores costs it can actually deteriorate solution quality if used within the hill climbing method, so in the remainder we do not consider this evaluator for hill climbing. Tables 5 and 6 show the results of the second experiment.

Table 5: Average costs after hill climbing.

	set $A1\|w=5$		set $A2\|w=6$		set $A3\|w=8$	
constructor→ hill climbing↓	OS^*	FS^*	OS^*	FS^*	OS^*	FS^*
OS	**792**	795	1173	**1170**	1540	**1538**
FS	815	814	1229	1219	1619	1611
	set $B1\|w=3$		set $B2\|w=5$		set $B3\|w=7$	
constructor→ hill climbing↓	OS^*	FS^*	OS^*	FS^*	OS^*	FS^*
OS	**684**	686	**1009**	1014	**1332**	**1332**
FS	717	720	1071	1070	1423	1422

The average cost observed for the 100 instances of each instance set is presented in Table 5. The results show that using evaluator OS during hill climbing clearly outperforms evaluator FS no matter whether OS^* or FS^* are used for construction of an initial feasible solution. This confirms that, during hill climbing, an evaluator focusing on assessing the exact cost of solutions should be used to achieve high solution quality.

Table 6: Average computation time *cpu* (in seconds) for construction and hill climbing.

	set $A1\|w=5$				set $A2\|w=6$				set $A3\|w=8$			
constructor→ hill climbing↓	OS^*	$FCOS^*$	$BFCOS^*$	FS^*	OS^*	$FCOS^*$	$BFCOS^*$	FS^*	OS^*	$FCOS^*$	$BFCOS^*$	FS^*
OS	15	14	13	13	56	52	51	51	147	139	138	133
$FCOS$	5	4	3	3	14	10	10	9	34	25	25	24
$BFCOS$	4	3	3	3	12	8	8	7	28	20	19	18
FS	2	1	1	<1	5	2	1	<1	12	3	3	1
	set $B1\|w=3$				set $B2\|w=5$				set $B3\|w=7$			
constructor→ hill climbing↓	OS^*	$FCOS^*$	$BFCOS^*$	FS^*	OS^*	$FCOS^*$	$BFCOS^*$	FS^*	OS^*	$FCOS^*$	$BFCOS^*$	FS^*
OS	16	15	15	14	64	61	60	57	181	175	172	174
$FCOS$	5	4	4	3	18	15	15	14	49	41	41	41
$BFCOS$	4	3	3	3	14	11	11	10	37	30	29	29
FS	2	1	1	<1	5	2	2	<1	12	4	4	1

Table 6 shows the *cpu* times (measured in seconds) required for the second experiment. In this table we also show results for evaluators $FCOS$ and $BFCOS$ and their corresponding adaptations $FCOS^*$ and $BFCOS^*$ which use the MST evaluator as fallback during construction. Table 6 shows that the overall computational effort is significantly smaller if evaluators $FCOS$ and $BFCOS$ are used instead of OS. The reason for this speed up is that feasibility of a work

plan can be checked much faster using the check() function presented in Section 4.4 compared to using an LP solver for the scheduling problem. As many infeasible work plans may be encountered during the search, this reduction in computational effort compensates for the additional effort required for the cases where feasible schedules exist. Evaluator $BFCOS$ appears to be yet a little faster than evaluator $FCOS$, however, obviously evaluator FS outperforms the others if very fast computation times must be achieved. It must also be noted that the choice of the evaluator used during hill climbing has a much higher impact on computation times than the choice of the evaluator used during construction.

5.4. Results for the Large Neighborhood Search

To evaluate the performance of the proposed Large Neighborhood Search we conduct a third experiment. For all tests in this experiment evaluator $BFCOS^*$ is used for constructing the initial solution for LNS and $BFCOS$ is used for improving this solution by the hill climber. This decision is drawn from our previous experiments where these evaluators achieved most of the best solutions and the lowest computation times among those evaluators that solve the scheduling subproblem to optimality.

In each LNS iteration, jobs are removed and reinserted and hill climbing is applied afterwards. Two evaluators are thus needed: one for reinserting jobs and one for hill climbing. Our previous experiments indicated that $BFCOS$ and FS appear to be good candidates for the evaluators used during LNS. Furthermore, we have conducted a series of pre-tests to determine appropriate

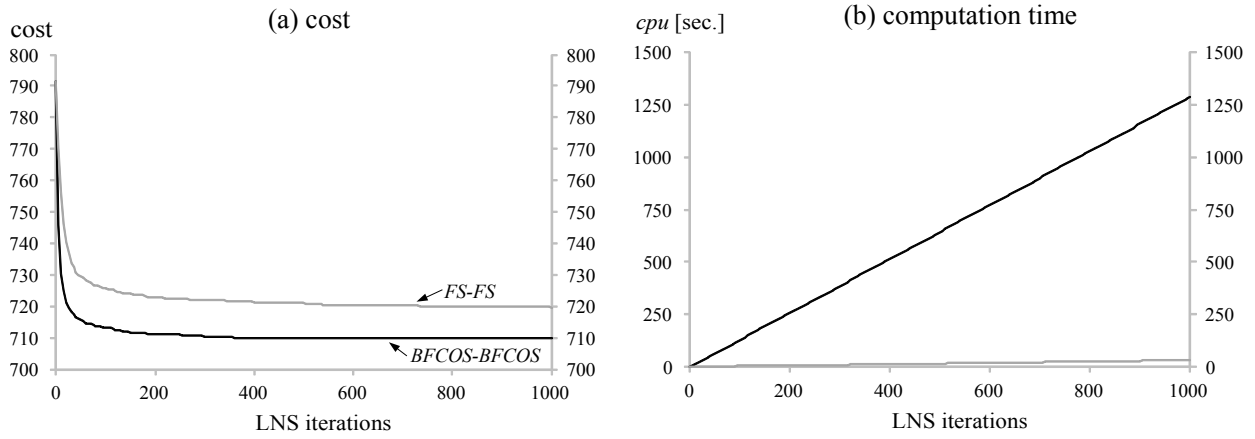

Figure 10: Convergence behavior of selected LNS configurations.

parameter values for the number of jobs to be removed and reinserted in each LNS iteration and the number of iterations to be performed by the LNS before terminating. Figure 10(a) illustrates the change in the solution cost and the required computation time when performing 1000 LNS iterations. The shown cost values and computation times are averages over the 100 instances of set $A1|w = 5$. We report these measures for two combinations of evaluators. In the first combination $BFCOS$ is used for both reinserting removed jobs and hill climbing whereas in the second combination FS is used for both purposes.

We see that solutions are quickly improved under both configurations. After a few hundred LNS iterations no further significant reduction in costs takes place. The final costs of $BFCOS$-$BFCOS$ and FS-FS are 710 and 720, respectively. Figure 10(b) shows the computation times that are needed for performing the 1000 iterations. We can observe that computation time per iteration remains fairly stable throughout the 1000 LNS iterations. For 1000 LNS iterations $BFCOS$-$BFCOS$ requires about 1300 seconds of computation time whereas FS-FS requires just about 30 seconds. Figures 11(a) and (b) illustrate the cost and computation times of both LNS configurations when changing the number of jobs k that are removed in each LNS iteration. We observe that too low and too high values of k lead to solutions of poor quality, whereas moderate values $k = 4$ to $k = 6$ produce solutions of lowest cost. Computation times are tremendously higher for $BFCOS$-$BFCOS$ than for FS-FS under any value of k. The same effects have been observed in further pre-tests for the other instance sets. We conclude from the results of Fig. 10 that 500 it-

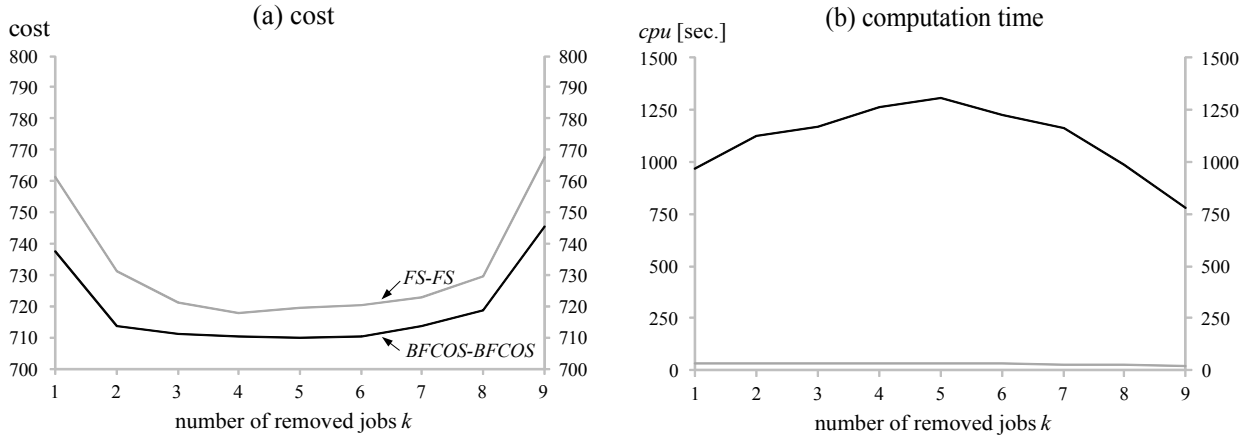

Figure 11: Impact of varied number of removed jobs.

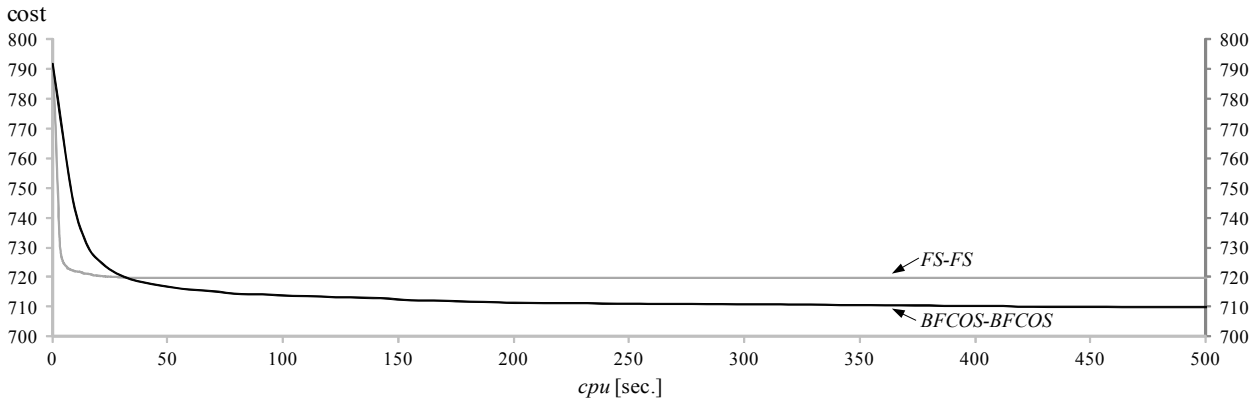

Figure 12: Cost vs. computation time of selected LNS configurations.

erations is an appropriate termination criterion for LNS and from Fig. 11 that $k = 5$ is a reasonable setting for removing jobs.

Our previous experiment indicated that $BFCOS$-$BFCOS$ not only achieved best solution quality but also brings a larger improvement than FS-FS for the first iterations in which costs are drastically reduced. However, if we compare both LNS configurations with respect to the improvement achieved per second of computation time, we observe that FS-FS can initially improve solution quality faster than $BFCOS$-$BFCOS$, see Fig. 12. Thus, if very little time is available for optimization FS-FS may be a better choice than $BFCOS$-$BFCOS$. This situation could occur in interactive optimization where a human planner and a computer system collaboratively search for a solution (see e.g. Goel & Gruhn, 2006) or if due to a failure in the electricity system a repair job must be performed immediately and the work plans of some workers must be revised quickly. In our further experiments we assume that very fast computation times are not required and the goal is to find solutions of high quality within reasonable time.

Tables 7 and 8 report the average cost and computation times observed for all six instance sets and combinations of $BFCOS$ and FS within LNS. The results show that LNS achieves another reduction in cost of up to 14% compared with the locally improved initial solutions of Table 5. Our experiments indicate that $BFCOS$-$BFCOS$ is the best combination for all instance sets and that FS leads to deteriorated solutions in all instance sets if it is used for construction and/or hill climbing. As most of the improvement is achieved within the first minute of the method an additional time limit could be used within the LNS if faster computation times are required.

Table 7: Average costs achieved by LNS configurations.

	set $A1\|w=5$		set $A2\|w=6$		set $A3\|w=8$		set $B1\|w=3$		set $B2\|w=5$		set $B3\|w=7$	
reinsertion→ / *hill climbing↓*	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS
$BFCOS$	**710**	714	**1034**	1045	**1354**	1379	**652**	653	**948**	963	**1250**	1280
FS	715	721	1049	1058	1400	1422	659	661	983	994	1319	1327

Table 8: Average computation time cpu (in seconds) required by LNS configurations.

	set $A1\|w=5$		set $A2\|w=6$		set $A3\|w=8$		set $B1\|w=3$		set $B2\|w=5$		set $B3\|w=7$	
reinsertion→ / *hill climbing↓*	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS	$BFCOS$	FS
$BFCOS$	642	440	1354	1021	2313	2119	343	209	984	905	2102	2477
FS	311	18	565	48	851	108	222	14	510	49	877	120

Table 9: Average results achieved by CPLEX.

	set $A1\|w=5$	set $A2\|w=6$	set $A3\|w=8$	set $B1\|w=3$	set $B2\|w=5$	set $B3\|w=7$
nmbr. feas. sol.:	100	61	7	26	1	1
avg. costs:	705	1049	1409	672	950	1706
avg. LB:	637	875	1118	524	750	972
avg. cpu:	36000	36000	36000	36000	36000	36000

In the previous experiments we considered sets of 100 test instances to avoid any bias resulting from the individual characteristics of the instances used. In the following experiment we furthermore assess the bias resulting from the inherent randomness of the method and the choice of random seeds. We have rerun the best performing LNS configuration $BFCOS$-$BFCOS$ with ten different random seeds for each of the instances. From the results, we observe that the average solution cost of the ten repetitions are within 0.1% of the cost reported in Table 7. We conclude that the random bias is negligibly and that the average cost obtained by a single test run provides a precise estimate on the performance of the heuristic.

To further assess the heuristic, we also applied the MIP solver of CPLEX to the six sets of 100 instances each. To aid CPLEX in solving the problem we added additional cuts which are described in Goel & Meisel (2013). In our experiments CPLEX was given a runtime of ten hours per instance and the test was conducted on a symmetric multiprocessing (SMP)-cluster equipped with 2.6 GhZ AMD Opteron CPUs. Aggregated results for all instance sets are shown in Table 9. The results reveal that CPLEX delivers 100 feasible solutions only for set $A1|w = 5$. However,

none of the instances is solved to optimality within the given runtime. The solutions found within ten hours show average cost of 705, which is a saving of merely 1% compared to the solutions returned by LNS within just about ten minutes. For all other sets, CPLEX does not succeed in finding a feasible solution for each of the 100 instances. Furthermore, the average cost of those solutions that are found is worse than the cost of solutions delivered by $BFCOS$-$BFCOS$, as can be seen when comparing Tables 7 and 9. The lower bounds returned by CPLEX can be used for further assessing the quality of the heuristic solutions. We observe average gaps for the best solutions found by LNS that range from 9.6% (set $A1$) to 20.2% (set $B3$). Hence, there may exist potential for further improvement of the heuristic solutions but it is also possible that CPLEX simply failed closing the gap as it is generally struggling with solving these instances.

Finally, we investigate how our approach behaves if a planner prefers objective weights that differ from the so far used $\lambda = 0.5$. For this experiment we compute a lower bound exploiting the 5-task structure of the jobs and the precedence relations (see Goel & Meisel, 2013). From solving the six instance sets with LNS configuration $BFCOS$-$BFCOS$, we observe that the downtimes approximate the lower bounds with increasing values of λ. For $\lambda = 0.95$ the largest relative gap is observed for $B1|w = 3$ with about 5%. For all other sets, the gap ranges between 1% and 3%. This confirms that the solutions produced by LNS are of very good quality and that they meet the important requirement of short downtimes in electricity network maintenance.

Note that detailed results for all test instances including the lower and upper bounds obtained by CPLEX, the best solutions found by LNS, and details on the final sensitivity analysis are provided in Goel & Meisel (2013).

6. Conclusions

In this paper, we study the operational problem occurring when a set of maintenance jobs in an electricity network must be performed by a given set of workers who must travel between locations associated to the different tasks belonging to a maintenance job. We model the problem as a mixed-integer program and show how high quality solutions of the problem can be obtained quickly. A Large Neighborhood Search meta-heuristic is presented which combines the heuristic generation of work plans with approximate and exact methods for scheduling of maintenance tasks. From

experiments on a large set of instances derived from a real network, we observed that best results are obtained when a linear programming based scheduling method combined with a heuristic feasibility check is used. The planning approach allows to generate work plans for maintenance workers that show very low downtimes and travel costs. The method can be parameterized by the planner to strive for the desired goal. The approach can also be used for determining the number of workers required to perform a given set of maintenance jobs.

References

Budai, G., Dekker, R., & Nicolai, R. (2008). Maintenance and production: A review of planning models. In K. Kobbacy, & D. Murthy (Eds.), *Complex System Maintenance Handbook* Springer Series in Reliability Engineering (pp. 321–344). Springer.

Budai, G., Huisman, D., & Dekker, R. (2006). Scheduling preventive railway maintenance activities. *Journal of the Operations Research Society*, *57*, 1035–1044.

Bundesnetzagentur (2012). SAIDI-Values. www.bundesnetzagentur.de/cln_1912/DE/Sachgebiete/ElektrizitaetGas/Sonderthemen/SAIDIWertStrom2010/SAIDIWertStrom2010_Basepage.html, accessed 27.09.2012.

Cordeau, J., Laporte, G., Pasin, F., & Ropke, S. (2010). Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, *13*, 393–409.

Cowling, P., Colledge, N., Dahal, K., & Remde, S. (2006). The trade off between diversity and quality for multi-objective workforce scheduling. In *Lecture Notes in Computer Science 3906* (pp. 13–24).

Dohn, A., Kolind, E., & Clausen, J. (2009). The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, *36*, 1145–1157.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, *46*, 297–316.

Goel, A., & Gruhn, V. (2006). Collaborative dispatching of commercial vehicles. In *Proceedings of the 2nd IEEE International Conference on Cybernetics & Intelligent Systems* (pp. 115–120).

Goel, A., & Gruhn, V. (2008). A general vehicle routing problem. *European Journal of Operational Research*, *191*, 650–660.

Goel, A., Gruhn, V., & Richter, T. (2010). Mobile workforce scheduling problem with multitask-processes. In S. Rinderle-Ma, S. Sadiq, & F. Leymann (Eds.), *Lecture Notes in Business Information Processing 43* (pp. 81–91). Springer.

Goel, A., & Meisel, F. (2013). Supplementary material for 'Workforce Routing and Scheduling for Electricity Network Maintenance with Downtime Minimization'. http://prodlog.wiwi.uni-halle.de/electricity-maintenance.

Gonçalves, J. F., Mendes, J. J. M., & Resende, M. G. C. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, *189*, 1171 – 1190.

IBM (2012). ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, accessed on 27.09.2012.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, & J. W. Thatcher (Eds.), *Complexity of Computer Computations* (pp. 85–103). New York: Plenum.

Kim, B.-I., Koo, J., & Park, J. (2010). The combined manpower-vehicle routing problem for multi-staged services. *Expert Systems with Applications*, *37*, 8424–8431.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, *90*, 320–333.

Krüger, D., & Scholl, A. (2009). A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, *197*, 492–508.

Li, Y., Lim, A., & Rodrigues, B. (2005). Manpower allocation with time windows and job-teaming constraints. *Naval Research Logistics*, *52*, 302–311.

Maniezzo, V., Stützle, T., & Voß, S. (Eds.) (2009). *Matheuristics: Hybridizing metaheuristics and mathematical programming* volume 10 of *Annals of Information Systems*. Springer.

Parragh, S., Doerner, K., & Hartl, R. (2008a). A survey on pickup and delivery problems - part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, *58*, 21–51.

Parragh, S., Doerner, K., & Hartl, R. (2008b). A survey on pickup and delivery problems - part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, *58*, 81–117.

Peng, F., Kang, S., Li, X., Ouyang, Y., Somani, K., & Acharya, D. (2011). A heuristic approach to the railroad track maintenance scheduling problem. *Computer-Aided Civil and Infrastructure Engineering*, *26*, 129–145.

Percy, D. F. (2008). Preventive maintenance models for complex systems. In K. Kobbacy, & D. Murthy (Eds.), *Complex Systems Maintenance Handbook* Springer Series in Reliability Engineering (pp. 179–207). Springer.

Savelsbergh, M., & Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, *29*, 17–30.

Shafiee, M., & Chukova, S. (2013). Maintenance models in warranty: A literature review. *European Journal of Operational Research*, *(in press)*.

Shaw, P. (1997). *A new local search algorithm providing high quality solutions to vehicle routing problems*. Technical Report APES Group, Department of Computer Science, University of Strathclyde Scotland.

Toth, P., & Vigo, D. (2002). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

Van den Bergh, J., Beliën, J., Bruecker, P. D., Demeulemeester, E., & Boeck, L. D. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*, 367 – 385.