Efficient Consideration of Soft Time Windows in a Large Neighborhood Search for the Districting and Routing Problem for Security Control[⋆]

Bong-Min Kim[1,2], Christian Kloimüllner[1], and Günther R. Raidl[1]

[1] Institute of Computer Graphics and Algorithms, TU Wien
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{kloimuellner|raidl}@ac.tuwien.ac.at
[2] Research Industrial Systems Engineering
Concorde Business Park F, 2320 Schwechat, Austria
bong-min.kim@rise-world.com

Abstract. For many companies it is important to protect their physical and intellectual property in an efficient and economically viable manner. Thus, specialized security companies are delegated to guard private and public property. These companies have to control a typically large number of buildings, which is usually done by teams of security guards patrolling different sets of buildings. Each building has to be visited several times within given time windows and tours to patrol these buildings are planned over a certain number of periods (days). This problem is regarded as the *Districting and Routing Problem for Security Control*. Investigations have shown that small time window violations do not really matter much in practice but can drastically improve solution quality. When softening time windows of the original problem, a new subproblem arises where the minimum time window penalty for a given set of districts has to be found for each considered candidate route: What are optimal times for the individual visits of objects that minimize the overall penalty for time window violations? We call this *Optimal Arrival Time Problem*. In this paper, we investigate this subproblem in particular and first give an exact solution approach based on *linear programming*. As this method is quite time-consuming we further propose a heuristic approach based on greedy methods in combination with dynamic programming. The whole mechanism is embedded in a large neighborhood search (LNS) to seek for solutions having minimum time window violations. Results show that using the proposed heuristic method for determining almost optimal starting times is much faster, allowing substantially more LNS iterations yielding in the end better overall solutions.

Keywords: Districting and Routing Problem for Security Control · Vehicle Routing Problem · Soft Time Windows · Dynamic Programming · Linear Programming

[⋆] We thank Günter Kiechle and Fritz Payr from CAPLAS GmbH for the collaboration on this topic. This work is supported by the Austrian Research Promotion Agency (FFG) under contracts 856215 and 849028.

1 Introduction

Theft and vandalism are a big and growing problem for many private and public companies. Thus, companies need to surveil their property, although permanent surveillance typically is not possible due to limited financial resources. Security companies, which are specialized experts, are therefore frequently engaged with observing the properties of these companies.

To minimize financial expenditures, objects are irregularly visited multiple times per day by security guards instead of dedicating a single security guard to a single object. Security guards have the duty of observing a particular set of objects. The number and times when these objects have to be visited may differ in each considered period. The problem of planning the districting and individual routes for performing the visits has been introduced by Prischink et al. [8] and is called *Districting and Routing Problem for Security Control* (DRPSC).

The previously proposed approach considers time windows in a strict sense. In practice, however, small time window violations typically do not matter much, and a larger flexibility in respect to them often allows substantially better solutions. In this paper, we consider the *Districting and Routing Problem for Security Control with Soft Time Windows* (DRPSC-STW). Soft time windows may be violated to some degree, and their violation is considered in the objective function by penalty terms. In this context, the subproblem of determining optimal visiting times for a given candidate tour so that the total time window penalty is minimized arises. We call this problem *Optimal Arrival Time Problem* (OATP).

To classify the DRPSC-STW in context of the vehicle routing literature, one can see it as a *periodic vehicle routing problem with soft time windows* with additional constraints concerning separation time and maximum tour duration, where objects may have to be visited multiple times in each period. Separation-time constraints are a minimum time difference between two consecutive visits of the same object in a tour. Moreover, each tour for every district and period must not exceed a given maximum tour duration.

In this work, we primarily focus on the OATP and how it can be effectively solved. To this end we propose an approach based on *linear programming* (LP) and a faster heuristic approach using greedy techniques and dynamic programming. These mechanisms are embedded in a large neighborhood search (LNS) [6].

The paper is structured as follows. Related work is given in Section 2 and the formal problem definition is stated in Section 3. Subsequently, we describe the OATP in Section 4 where we also introduce the LP approach. Then, in Section 5, the efficient hybrid heuristic for solving the OATP is introduced and in Section 6 the LNS metaheuristic for approaching the DRPSC-STW is proposed. Experiments are performed in Section 7 and, finally, a conclusion as well as an outlook for future work is given in Section 8.

2 Related Work

Prischink et al. [8] introduce the DRPSC and propose two construction heuristics as well as a sophisticated *district elimination algorithm* for the districting part of the problem. In the district elimination algorithm they iteratively eliminate a district, put the objects of these districts in a so called *ejection pool* and then try to insert the objects of this ejection pool again into the set of available districts. We adopt this idea/mechanism for developing a *destroy and recreate* neighborhood inside our LNS. We, thus, remove the objects of two, uniformly at random selected, districts and put them into a so called *ejection pool* but do not delete the districts from which we removed the objects. Then, we execute a single run/step of the proposed district elimination algorithm which tries to reassign the objects in the ejection pool to the available districts, and let the algorithm terminate if the ejection pool is empty at the end of this single iteration.

As the focus of our current work lies on the extension to soft time windows, we also put our attention here on previous work dealing with them. Although much more work is done on problem types with hard time windows, there already exists a significant number of works which introduce efficient methods to effectively handle soft time window constraints.

Ibaraki et al. [4] proposed a dynamic programming (DP) based approach to determine optimal starting times in conjunction with soft time windows which is applicable to a wider range of routing and scheduling applications. The total penalty incurred by time window violations is minimized. Compared to our approach they consider more general piecewise-linear penalty functions. Unfortunately, their approach is not directly applicable in our context as we have to additionally consider minimum separation times between visits of the same objects (i.e., objects can only be visited again if a minimum separation time between two consecutive visits is considered) and a maximum tour length. However, we will show later how this efficient DP method can nevertheless be utilized to some degree in our case.

Hashimoto et al. [3] extended the work of Ibaraki et al. to also consider flexible traveling times, which are also penalized if violated. They show, however, that the problem becomes NP-hard in case.

Taillard et al. [9] solve the vehicle routing problem with soft time windows by using tabu search. They do not consider any penalties for arriving too early but introduce a “lateness penalty” into the objective function. This penalty value is weighted by a given factor and the problem can be transformed into the vehicle routing problem with hard time windows by setting the weight factor to ∞.

Another work which shows the efficiency of applying DP for solving problems with soft time windows is by Ioachim et al. [5]. They solve the shortest path problem with time windows and linear node costs, where the linear node costs correspond to the modeling of soft time windows.

Fagerholt [2] published an approach for *ship scheduling with soft time windows*. He argues that by considering soft time windows, solution quality can be drastically improved and in practice small time window violations do not really matter. As in our work, a maximum allowed time window violation is used and earlier and later service is penalized. The approach can handle also non-convex penalty functions whereas in the literature most often only convex penalty functions are considered. The proposed solution approach uses a discretized time network in which nodes are duplicated according to possible start/arrival times. On the obtained shortest path network problem DP is applied for obtaining optimal arrival times.

To summarize related work, DP can frequently be an effective tool to determine optimal arrival/service times when considering soft time windows. Certain specificities of problems like maximal total tour duration and other constraints, however, frequently become an obstacle and prohibit the direct application of an efficient DP as the subproblem of determining optimal arrival times becomes NP hard. Nevertheless, DP may still be an important ingredient to deal with such situations in practice.

3 Problem Definition

In the DRPSC-STW we are given a set of objects $I = \{1, \ldots, n\}$ and a depot 0, which is the start and end of each route. Travel times among the objects and the depot are given by $t^{\text{travel}}_{i,i'} > 0$ for $i, i' \in I \cup \{0\}$. We assume the triangle inequality to hold among these travel times. For every object $i \in I$ we are given a (small) number of visits $S_i = \{i_1, \ldots, i_{|S_i|}\}$, and we are given a set of periods $P = \{1, \ldots, p\}$. As not all visits have to take place in every period, subsets $W_{i,j} \subset S$ contain the visits of object i requested in period j for all $i \in I,\ j \in P$. The depot is visited two times, namely at the start of the tour and at the end of the tour. To ease modeling we define 0_0 to be the departure from the depot at the beginning and 0_1 to be the arrival at the depot at the end.

Each visit $i_k \in S_i, i \in I$ is associated with a visit time $t^{\text{visit}}_{i_k}$ and a particular time window $T_{i_k} = [T^{\text{e}}_{i_k}, T^{\text{l}}_{i_k}]$ in which the whole visit should preferably take place, already including its visit time. Visits $i_k \in S_i$ of an object $i \in I$ have to be visited in the given order, i.e., visit k has to be performed before visit k' iff $k < k'$.

Time windows of the visits are now softened such that an earlier start or later finishing of the service at an object is allowed. The maximum allowed earliness and lateness are, however, restricted by Δ, yielding the hard time windows $T^{\text{h}}_{i_k} = [T^{\text{he}}_{i_k}, T^{\text{hl}}_{i_k}] = [T^{\text{e}}_{i_k} - \Delta, T^{\text{l}}_{i_k} + \Delta]$, which must not be violated in any feasible solution.

An additional important requirement in the context of our security application is that any two successive visits $i_k, i_{k+1} \in W_{i,j}$ of the same object $i \in I$ must be separated by a *minimum separation time* t^{sep}. Obviously, visiting an object twice without a significant time inbetween would not make much sense. The maximum duration of any tour is given by $t^{\max}$.

Solutions to the DRPSC-STW are given by a tuple (D, τ, a) where $D = \{D_1, \ldots, D_m\}$ is the partitioning of objects into districts, $\tau = (\tau_{r,j})_{r=1,\ldots,m,\ j \in P}$ are the routes for each district and period, and a denotes the respective arrival times. Each tour $\tau_{r,j} = (\tau_{r,j,0}, \ldots, \tau_{r,j,l_{r,j}+1})$ with $l_{r,j} = \sum_{i \in D_r} |W_{i,j}|$ starts and ends at the depot, i.e., $\tau_{r,j,0} = 0_0$ and $\tau_{r,j,l_{r,j}+1} = 0_1$, $\forall r = 1, \ldots, m, j \in P$, and performs each visit in the respective ordering of the sequence. Each visit of a tour $\tau_{r,j,u}$ has to be associated with a specific arrival time $a_{r,j,u}$ and thus, $a = (a_{r,j,u})_{r=1,\ldots,m,\ j=1,\ldots,p,\ u=1,\ldots l_{r,j}+1}$. An object always is immediately serviced after arrival but waiting is possible before leaving the object. A tour is feasible, if all visit, travel, and separation times are considered, each visit is performed at least within its hard time window and the total tour duration does not exceed $t^{\max}$.

While in our previous work [7] the primary objective was to minimize the number of districts (m), we consider this number now as pre-specified. For example, it can be obtained in a first optimization round by our previous method based on the hard time windows only. Now, in the DRPSC-STW, our objective is to minimize the total penalty incurred by all time window violations, which is

$$\min \sum_{r=1}^{m} \sum_{j \in P} \sum_{u=1}^{l_{r,j}} \omega_{r,j,u} \tag{1}$$

with

$$\omega_{r,j,u} = \begin{cases} T^{\mathrm{e}}_{i_k} - a_{r,j,u} & \text{if } a_{r,j,u} < T^{\mathrm{e}}_{i_k} \\ a_{r,j,u} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} & \text{if } a_{r,j,u} + t^{\mathrm{visit}}_{i_k} > T^{\mathrm{l}}_{i_k} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

4 Optimal Arrival Time Problem

When approaching the DRPSC-STW with an LNS in Section 6, we will have to solve for each tour in each period of each candidate solution the following subproblem: Given a candidate tour $\tau_{r,j} = (\tau_{r,j,0}, \ldots, \tau_{r,j,l_{r,j}+1})$ for some district $r = 1, \ldots, m$ and period $j = 1, \ldots, p$, what are feasible arrival times $a_{r,j,u}$ for the visits $u = 1, \ldots, l_{r,j} + 1$ minimizing $\sum_{u=1}^{l_{r,j}} \omega_{r,j,u}$? Remember that the solution must obey the minimum separation time t^{sep} between any two successive visits of the same object and the maximum tour duration $t^{\max}$. We call this subproblem *Optimal Arrival Time Problem* (OATP).

As we consider in the OATP always only one specific tour $\tau_{r,j}$, i.e., r and j are known and constant, we omit these indices in the following for simplicity wherever this is unambiguous. In particular, we write τ for the current tour, l for the tour's length, τ_h for the h-th visit, a_h for the respective arrival time, and ω_h for the respective time window penalty. Moreover, we introduce some further notations and definitions used in the next sections. Let us more generally define the time window penalty function $\rho_h(t)$ for visit $\tau_h = i_k$ when arriving at time t as the following piecewise linear function, see also Figure 1:

$$\rho_h(t) = \begin{cases} \infty & \text{if } t < T^{\mathrm{e}}_{i_k} - \Delta \\ T^{\mathrm{e}}_{i_k} - t & \text{if } T^{\mathrm{e}}_{i_k} - \Delta \le t < T^{\mathrm{e}}_{i_k} \\ t + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} & \text{if } T^{\mathrm{l}}_{i_k} < t + t^{\mathrm{visit}}_{i_k} \le T^{\mathrm{l}}_{i_k} + \Delta \\ \infty & \text{if } t > T^{\mathrm{l}}_{i_k} + \Delta \\ 0 & \text{otherwise.} \end{cases}$$

Let $V = \{i_k \mid i \in D_r,\ i_k \in W_{i,j}\}$ be the set of all object visits in the current

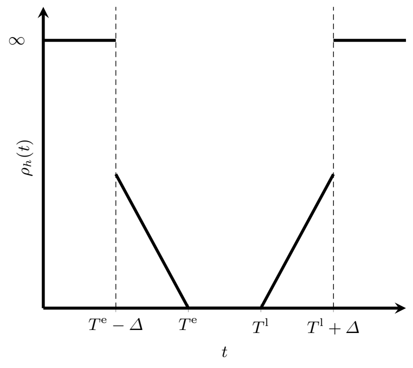


Fig. 1. The time window penalty function $\rho_h(t)$.

tour. We define the auxiliary function $\kappa : V \mapsto D_r$ mapping visit $i_k \in V$ to its corresponding object $i \in D_r$, and function $\sigma(h)$ which finds the nearest successor index h' of the visit $\tau_{h'}$ with $h < h' \leq l$ and $\kappa(\tau_h) = \kappa(\tau_{h'})$ if such a successive visit of the same object exists; otherwise $\sigma(h)$ returns -1. Correspondingly, function $\sigma^{-1}(h)$ returns the nearest predecessor index h' of the visit $\tau_{h'}$ with $1 \leq h' < h$ and $\kappa(\tau_h) = \kappa(\tau_{h'})$ if such a predecessor exists and -1 otherwise. For convenience, we also define $\zeta_h = t^{\mathrm{visit}}_{\tau_{r,j,h}} + t^{\mathrm{travel}}_{\kappa(\tau_{r,j,h}),\kappa(\tau_{r,j,h+1})}$ as the sum of the visiting time of the hth visit and the travel time from the hth visit to the $(h+1)$st visit.

4.1 Lower and Upper Bounds for Arrival Times

We compute lower and upper bounds for each visit's arrival time by determining routes in which we perform each visit as early as possible and as late as possible. For determining the earliest arrival time at the first visit we have to consider the maximum of the travel time from the depot to the first visit and the earliest possible time of the first visit's hard time window. The earliest possible arrival time for all other visits $h = 1, \ldots, l$ can be computed recursively by considering the dependency on the previous visit $h-1$, i.e., the visit time and travel time to the current visit h, the beginning of the hard time window $T^{\mathrm{e}}_{\tau_h} - \Delta$ of the current visit h, and the separation time from a possibly existing previous visit of the same object $\sigma^{-1}(h)$ in the tour. This yields:

$$a^{\mathrm{earliest}}_h = \begin{cases} -\infty & \text{if } h < 0 \\ T^{\mathrm{e}}_{0_0} & \text{if } h = 0 \\ \max \left\{ a^{\mathrm{earliest}}_{h-1} + t^{\mathrm{visit}}_{\tau_{h-1}} + t^{\mathrm{travel}}_{\tau_{h-1},\tau_h}, T^{\mathrm{e}}_{\tau_h} - \Delta, a^{\mathrm{earliest}}_{\sigma^{-1}(h)} + t^{\mathrm{sep}} \right\} & \text{if } h > 0 \end{cases}$$

When scheduling a latest tour the last visit of the tour has to be scheduled before arriving at the depot where also the travel time to the depot has to be considered, but on the other hand we have to also consider the end of the hard time window $T^{\mathrm{l}}_{\tau_l} + \Delta$ of the last visit. For all other visits we can compute their latest possible arrival time by considering the next visit's arrival time, the travel time to the next visit, and the visit time at the current visit, the end of the hard time window of the current visit, i.e., $T^{\mathrm{l}}_{\tau_l} + \Delta$, and the separation time by considering a possibly existing successive visit $\sigma(h)$ of the same object where $\kappa(\tau_h) = \kappa(\tau_{h'})$ with $h < h'$:

$$a^{\mathrm{latest}}_h = \begin{cases} \infty & \text{if } h < 0 \\ T^{\mathrm{l}}_{0_1} & \text{if } h = l+1 \\ \min\left\{a^{\mathrm{latest}}_{h+1} - t^{\mathrm{visit}}_{\tau_h} - t^{\mathrm{travel}}_{\tau_h,\tau_{h+1}}, T^{\mathrm{l}}_{\tau_h} + \Delta, a^{\mathrm{latest}}_{\sigma(h)} - t^{\mathrm{sep}}\right\} & \text{if } 0 \leq h \leq l \end{cases}$$

If for some h, $a^{\mathrm{earliest}}_h > a^{\mathrm{latest}}_h$, we immediately terminate as this OATP instance, i.e., underlying route, cannot have a feasible solution.

4.2 Linear Programming Model

The OATP is not an NP-hard optimization problem. We can solve it exactly by means of linear programming (LP) as we show in the following.

Variables a_{i_k} represent the arrival time of the k-th visit of object i, variables $p^{\mathrm{e}}_{i_k}$ are used to compute the penalty when starting the service of visit i_k too early, and variables $p^{\mathrm{l}}_{i_k}$ are used for the penalty when finishing the service of visit i_k too late. The LP is defined as follows:

$$\begin{aligned}
\min \quad & \sum_{i_k \in V} p^{\mathrm{e}}_{i_k} + p^{\mathrm{l}}_{i_k} && (3)\\
\text{s.t.} \quad & t^{\mathrm{travel}}_{0,\kappa(a_{\tau_1})} + a_{\tau_l} + t^{\mathrm{visit}}_{\tau_l} + t^{\mathrm{travel}}_{\kappa(\tau_l),0} - a_{\tau_1} + \leq t^{\max} && (4)\\
& a_{\tau_1} \geq t^{\mathrm{travel}}_{0,a_{\tau_1}} + T^{\mathrm{e}}_{0_0} && (5)\\
& a_{\tau_l} + t^{\mathrm{visit}}_{\tau_{a_l}} + t^{\mathrm{travel}}_{\kappa(\tau_l),0} \leq T^{\mathrm{l}}_{0_1} && (6)\\
& a_{\tau_i} \geq a_{\tau_{i-1}} + t^{\mathrm{visit}}_{a_{\tau_{i-1}}} + t^{\mathrm{travel}}_{\kappa(\tau_{i-1}),\kappa(\tau_i)} & \forall \tau_i \in \tau,\ i = 2,\ldots,l \quad & (7)\\
& a_{i_k} \geq a_{i_{k'}} + t^{\mathrm{visit}}_{i_{k'}} + t^{\mathrm{sep}} & \forall i_k, i_{k'} \in V,\ k > k' \quad & (8)\\
& T^{\mathrm{e}}_{i_k} - \Delta \leq a_{i_k} \leq T^{\mathrm{l}}_{i_k} + \Delta - t^{\mathrm{visit}}_{i_k} & \forall i_k \in V \quad & (9)\\
& p^{\mathrm{e}}_{i_k} \geq T^{\mathrm{e}}_{i_k} - a_{i_k} & \forall i_k \in V \quad & (10)\\
& p^{\mathrm{l}}_{i_k} \geq a_{i_k} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} & \forall i_k \in V \quad & (11)\\
& a_{i_k}, p^{\mathrm{e}}_{i_k}, p^{\mathrm{l}}_{i_k} \geq 0 & \forall i_k \in V \quad & (12)
\end{aligned}$$

Objective function (3) minimizes the total penalty incurred by too late or too early arrival times of visits. Inequality (4) ensures that the makespan of the tour does not exceed the maximum allowed duration $t^{\max}$. Otherwise, the given visit order would be infeasible. Inequality (5) models the travel time from the depot to the first visit of the given order, i.e., the first visit can only be started after traveling from the depot to this visit. Inequality (6) specifies that the tour has

Algorithm 1 Hybrid Heuristic for OATP

```
1: Input: Tour $\tau$
2: if not Feasible($\tau$) then
3:     return $\infty$
4: end if
5: if GreedyHeuristic($\tau$) $= 0$ then
6:     return 0
7: end if
8: return DPBasedHeuristic($\tau$)
```

to end latest at the end of the time window of the second visit of the depot. Inequalities (7) ensure that all travel times between consecutive object visits and visit times are respected. Inequalities (8) guarantee the minimum separation time between two consecutive visits of the same object. Inequalities (9) ensure consideration of the hard time windows. The penalty values are computed by inequalities (10) and (11). If a visit is scheduled too early, then $T^{\mathrm{e}}_{i_k} - a_{i_k} > 0$ and an early penalty is incurred. Obviously, if the earliness penalty $p^{\mathrm{e}}_{i_k} > 0$, then $a_{i_k} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} < 0$ and thus, $p^{\mathrm{l}}_{i_k} = 0$. This holds vice versa if the lateness penalty $p^{\mathrm{l}}_{i_k} > 0$. If a visit is scheduled within its time window $[T^{\mathrm{e}}_{i_k}, T^{\mathrm{l}}_{i_k}]$, then $p^{\mathrm{e}}_{i_k} = p^{\mathrm{l}}_{i_k} = 0$ as $T^{\mathrm{e}}_{i_k} - a_{i_k} \leq 0$ and $a_{i_k} + t^{\mathrm{visit}}_{i_k} - T^{\mathrm{l}}_{i_k} \leq 0$ and $p^{\mathrm{e}}_{i_k}, p^{\mathrm{l}}_{i_k} \geq 0$, $\forall i_k \in V$ according to equations (12).

5 Hybrid Heuristic for the OATP

While the above LP model can be solved in polynomial time, doing this many thousands of times within a metaheuristic for the DRPSC-STW for evaluating any new tour in any period of any district in any candidate solution is still a substantial bottleneck. We therefore consider a typically much faster heuristic approach in the following, which, as our experiments will show, still yields almost optimal solutions. We call this approach *Hybrid Heuristic* (HH) for the OATP as it is, in fact, a sequential combination of different individual components.

The overall approach is shown in Algorithm 1, and the individual components are described in detail in the subsequent sections. First, we show how to efficiently check the feasibility of a given instance (line 2), then, we apply a fast greedy heuristic which tries to solve the problem without penalties (line 5) using an earliest possible start time strategy. Finally, we apply an efficient DP-based heuristic to obtain a solution for the OATP.

5.1 Feasibility Check

The feasibility of a given tour, i.e., existence of feasible arrival times, can be efficiently checked by calculating the minimum tour duration and comparing it to $t^{\max}$. The minimum tour duration can be determined by fixing the arrival time at the depot to $a_{l+1}^{\text{earliest}}$ and calculating the latest arrival times recursively backwards:

$$a_h^{\text{ms}} = \begin{cases} \infty & \text{if } h < 0 \\ a_h^{\text{earliest}} & \text{if } h = l+1 \\ \min\left\{a_{h+1}^{\text{ms}} - t_{\tau_h}^{\text{visit}} - t_{\tau_h,\tau_{h+1}}^{\text{travel}}, T_{\tau_h}^{\text{l}} + \Delta, a_{\sigma(h)}^{\text{ms}} - t^{\text{sep}}\right\} & \text{if } 0 \le h \le l \end{cases}$$

The tour is feasible iff $a_{l+1}^{\text{ms}} - a_0^{\text{ms}} \le t^{\max}$ holds.

5.2 Greedy Heuristic

A fast heuristic for solving the OATP is a greedy strategy that starts each visit as early as possible without violating any soft time window. If this heuristic is successful, no penalty occurs and the obtained solution is optimal. We can formulate this approach as follows:

$$a_h^{\text{greedy}} = \begin{cases} -\infty & \text{if } h < 0 \\ \max\left\{T_{0_0}^{\text{e}} + t_{0,\kappa(\tau_1)}^{\text{travel}}, T_{\tau_h}^{\text{e}}\right\} & \text{if } h = 1 \\ \max\left\{a_{h-1}^{\text{greedy}} + t_{\tau_{h-1}}^{\text{visit}} + t_{\tau_{h-1},\tau_h}^{\text{travel}}, T_{\tau_h}^{\text{e}}, a_{\sigma^{-1}(h)}^{\text{greedy}} + t^{\text{sep}}\right\} & \text{if } h > 1 \end{cases}$$

If for some h, $a_h^{\text{greedy}} > a_h^{\text{latest}}$, then the greedy heuristic cannot solve this problem instance and terminates.

5.3 Efficiently Solving a Relaxation by Dynamic Programming

The greedy strategy is fast, works reasonably well, and frequently yields an optimal solution for easy instances. When the constraints become tighter, however, it often fails. Therefore, we finally use a second, more sophisticated heuristic based on the following considerations.

The required minimum separation times for visits of same objects make the OATP, in contrast to other problems aiming at finding arrival times introducing a minimum penalty, e.g. [4], inaccessible for an efficient exact DP approach. One would need to somehow consider also all objects' last visits when storing and reusing subproblem solutions in the DP recursion.

However, in a heuristic approach we can exploit an efficient DP for the relaxed variant of the OATP in which we remove the separation time constraints. We denote this relaxed OATP by OATP$^{\text{rel}}$. As will be shown in Section 5.4, we will modify our instance data before applying this DP in order to obtain a heuristic solution that is feasible for our original OATP.

To solve OATP$^{\text{rel}}$ we apply DP inspired by Ibaraki et al. [4]. In contrast to this former work, however, we consider a maximum tour duration.

Let $g_h(t, t_0)$ be the minimum sum of the penalty values for visits $\tau_0, \ldots, \tau_h$ under the condition that all of them are started no later than at time t and the depot is left earliest at time t_0 with $t - t_0 \leq t^{\max}$. Here we assume that $T^{\mathrm{e}}_{0_0} \leq t_0$. Then, $g_h(t, t_0)$ can be expressed recursively by:

$$g_0(t, t_0) = \begin{cases} \infty & \text{if } t < t_0 \\ 0 & \text{otherwise} \end{cases}$$
$$g_h(t, t_0) = \min_{a_h^{\text{earliest}} \leq t' \leq \min\{t, t_0 + t^{\max}\}} g_{h-1}(t' - \zeta_{h-1}, t_0) + \rho_h(t') \quad \text{if } h > 0$$

Here, we assume the minimum of an empty set or interval to be ∞. The overall minimum time penalty of the tour τ is then $\min_{a_0^{\text{earliest}} \leq t_0 \leq a_0^{\text{latest}}} g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$. Thus, solving OATP$^{\text{rel}}$ corresponds to finding a departure time t_0 from the depot which minimizes function $f^{\text{rel}} = g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$.

Let t_0 be the value for which $f^{\text{rel}} = g_h(T^{\mathrm{l}}_{0_1}, t_0)$ yields a minimum penalty. Optimal arrival times for the visits and the arrival time back at the depot can now be expressed by:

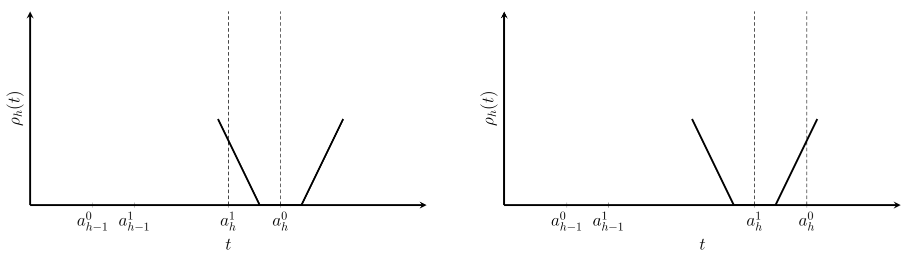
$$\begin{aligned} a^{\text{rel}}_{l+1} &= \operatorname*{arg\,min}_{T^{\mathrm{e}}_{0_0} \leq t \leq T^{\mathrm{l}}_{0_1}} g_{l+1}(t, t_0) \\ a^{\text{rel}}_{h} &= \operatorname*{arg\,min}_{T^{\mathrm{e}}_{0_0} \leq t \leq a^{\text{rel}}_{h+1} - \zeta_h} g_h(t, t_0) \quad \text{if } 0 \leq h \leq l \end{aligned} \tag{13}$$

Now, let us consider the task of efficiently computing $g_h(t, t_0)$ in more detail. Recall that our time window penalty function $\rho_h(t)$ is piecewise linear for all visits $\tau_1, \ldots, \tau_l$ and they have all the same shape as shown in Figure 1. Therefore, $g_h(t, t_0)$ is also piecewise linear. We store these piecewise linear functions of each recursion step of the DP algorithm in linked lists, whose components represent the intervals and the associated linear functions.

An upper bound for the total number of pieces in the penalty functions for all the visits $\tau_0, \ldots, \tau_{l+1}$ is $5l+2 = O(l)$. The computation of $g_{h-1}(t-\zeta_{h-1}, t_0)+\rho_h(t)$ and $g_h(t, t_0)$ from $g_{h-1}(t, t_0)$ and $\rho_h(t)$ can be achieved in $O(h)$ time, since the total number of pieces in $g_{h-1}(t, t_0)$ and $\rho_h(t)$ is $O(h)$. In order to calculate the function $g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$ for a given tour, we compute $g_h(t, t_0)$ for all $1 \leq h \leq l+1$. This can be done in $O(l^2)$ time.

Now that we know how to efficiently calculate the minimum time window penalty value for a given departure time from the depot t_0, we draw our attention to the problem of finding a best departure time such that the overall penalty value for a given tour is minimized. Formally, we want to minimize function $g'(t_0) = g_{l+1}(T^{\mathrm{l}}_{0_1}, t_0)$ on interval $t_0 \in [a_0^{\text{earliest}}, a_0^{\text{latest}}]$. Enumerating all possible t_0 values is obviously not a reasonable way to tackle this problem. Fortunately, there is a useful property of function $g'(t_0)$ which enables us to search more efficiently for its minimum.

Proposition 1. *Earliest optimal arrival times can only be delayed further when the depot departure time increases. More formally, let a_h^0 for $h = 0, \ldots, l+1$ be earliest optimal arrival times calculated by $g'(t_0)$ for some t_0 and a_h^1 for $h = 0, \ldots, l+1$ be the earliest optimal arrival times calculated by $g'(t'_0)$ for some t'_0. Then $t_0 \leq t'_0 \implies a_h^0 \leq a_h^1$ for $h = 0, \ldots, l+1$.*



(a) Case 1 (b) Case 2

Fig. 2. Visualization of the two case distinctions used in the proof of Proposition 1.

Proof. We show this with a proof by contradiction. Without loss of generality, suppose there is a visit h with $a_{h-1}^0 \leq a_{h-1}^1$ and $a_h^1 < a_h^0$. Let us consider two relevant cases in detail. Other cases can be refuted using similar arguments.

Case 1: Assume a_h^1 is scheduled earlier than a_h^0 and $a_h^1 < T_{\tau_h}^{\mathrm{e}}$, see Figure 2a. a_h^1 could only have been scheduled earlier than a_h^0 falling below $T_{\tau_h}^{\mathrm{e}}$ threshold if and only if one of its subsequent visits $\tau_{h+1}, \ldots, \tau_{l+1}$ was forced to start earlier. This can only happen if the arrival time constraint, where we have to be back at the depot, is more tightened. But this clearly cannot be the case here, since $t_0 + t^{\max} \leq t_0' + t^{\max}$. In other words, delaying the departure time at the depot also delays the arrival time constraint, when we have to be back at the depot.

Case 2: Assume a_h^1 is scheduled earlier than a_h^0 and $a_h^0 > T_{\tau_h}^{\mathrm{l}} - t_{\tau_h}^{\mathrm{visit}}$, see Figure 2b. Since $a_h^0 - a_{h-1}^0 > a_h^1 - a_{h-1}^1$, it is easy to see that a_h^0 can be moved further to the left without introducing more penalty. Therefore, a_h^0 cannot be the optimal start time for the visit h, since the T_h^{l} constraint violation caused by a_h^0 can be reduced further.

□

Proposition 2. $\forall t_0', t_0'' \mid g'(t_0') < g'(t_0''), t_0' < t_0'' \implies \forall t_0 \geq t_0'' : g'(t_0'') < g'(t_0)$.

Proof. Let $a_h^{\mathrm{earliest}'}$ for $h = 0, \ldots, l+1$ be the earliest possible arrival times when fixing t_0' as the departure time from the depot and $a_h^{\mathrm{earliest}''}$ for $h = 0, \ldots, l+1$ the earliest possible arrival times when fixing t_0'' as the departure time from the depot. Furthermore, we define a_h'' for $h = 0, \ldots, l+1$ to be earliest optimal arrival times calculated by $g'(t_0'')$.

We have shown that the earliest optimal arrival times can only be delayed further when postponing the departure time from the depot. Thus, the only way the overall penalty value can be increased is when pushing t_0 to the future causes more T^{l} threshold violations than what you can save by reducing T^{e} threshold violations.

More formally, if we have $g'(t_0') < g'(t_0'')$ with $t_0' < t_0''$, then there must exist $a_k^{\mathrm{earliest}''} > T_{\tau_k}^{l} - t_{\tau_k}^{\mathrm{visit}}$ with $a_k^{\mathrm{earliest}'} < a_k^{\mathrm{earliest}''}$ and $a_k^{\mathrm{earliest}''} = a_k''$ for some

Algorithm 2 Calculation of f^{rel}

```
Input: $a_0^{\text{earliest}}$, $a_0^{\text{latest}}$
 1: init: $a \leftarrow a_0^{\text{earliest}}$, $b \leftarrow a_0^{\text{latest}}$, $v_1 \leftarrow f^{\text{rel}} \leftarrow g'(a)$
 2: if $v_1 = 0$ or $v_1 = \infty$ or $\nabla g'(t) > 0$ then
 3:     return $v_1$
 4: end if
 5: while $b - a > \varepsilon$ do
 6:     $t \leftarrow a + \frac{b-a}{2}$
 7:     $v_2 \leftarrow g'(t)$
 8:     if $v_2 < f^{\text{rel}}$ then
 9:         $f^{\text{rel}} \leftarrow v_2$
10:     end if
11:     if $f^{\text{rel}} = 0$ or $v_1 = v_2$ then
12:         break
13:     end if
14:     if $v_2 = \infty$ or $\nabla g'(t) \leq 0$ then
15:         $a \leftarrow t$
16:     else
17:         $b \leftarrow t$
18:     end if
19:     $v_1 \leftarrow v_2$
20: end while
21: return $f^{\text{rel}}$
```

$k \in \{0, \ldots, l+1\}$. In other words, if the overall penalty value increases, then there are visits whose earliest possible arrival times are pushed furhter to the future exceeding T^{l} thresholds by t_0'' and their optimal arrival times are equal to earliest possible arrival times.

It is easy to see that once the earliest possible start time a_h^{earliest} starts to increase, it continues to increase strictly monotonically with an increasing departure time from the depot. Therefore, the overall penalties will increase strictly monotonically from t_0'' on with an increasing departure time from the depot until the solution becomes infeasible. □

These properties show that $g'(t_0)$ is in general a "U-shaped" function when disregarding all infeasible solutions yielding ∞, and we can use a bisection method to search efficiently for a minimum. The calculation of f^{rel} in this way is shown in Algorithm 2.

At each iteration step the middle point t of current search interval is sampled and we calculate an approximate subgradient $\nabla g'(t)$ of g' at t by $\nabla g'(t) = g'(t+\delta) - g'(t)$ where δ is a small constant value. If the subgradient $\nabla g'(t) > 0$, we know that t is in the strictly monotonically rising piece of g' and we continue our search in the left half. Otherwise the search continues in the right half. The bisection method proceeds until the search interval becomes smaller than some predetermined value ε.

5.4 DP-Based Heuristic for OATP

Obviously, OATP$^{\text{rel}}$ corresponds to the original OATP if there are no objects that are visited multiple times or $\sum_{i=h}^{\sigma(h)-1} \zeta_i \geq t^{\text{sep}}$ for $h = 1, \dots, l$ with $\sigma(h) \neq -1$. The main idea of our second heuristic is to increase the ζ_i values as necessary so that $\sum_{i=h}^{\sigma(h)-1} \zeta_i \geq t^{\text{sep}}$ holds for all $h = 1, \dots, l$ with $\sigma(h) \neq -1$. Then, when applying the DP, its solution will obviously fulfill the separation-time constraint.

Let visits τ_k and $\tau_{k'}$ with $k < k'$ and $\sum_{i=k}^{k'-1} \zeta_i < t^{\text{sep}}$ be two visits which take place at the same object. Then, one or more $\zeta_i \in \{\zeta_k, \dots, \zeta_{k'-1}\}$ must be extended so that $\sum_{i=k}^{l-1} \zeta_i = t^{\text{sep}}$. In order to decide which ζ_i we want to extend, we first calculate waiting times for all visits with earliest possible arrival times.

The waiting time at the visit τ_h is the amount of time we are forced to wait at the visit τ_{h-1} before we can travel to visit τ_h. Recall that we are forced to wait at visit τ_{h-1} if $a_{h-1} + \zeta_{h-1} < T^{\text{e}}_{\tau_h}$. Thus, the waiting times with earliest possible arrival times can be expressed as $w_h^{\text{earliest}} = \max\left\{0, a_h^{\text{earliest}} - a_{h-1}^{\text{earliest}} - \zeta_{h-1}\right\}$, $h = 1, \dots, l$. Using these waiting times as guidance, we extend the ζ_i value at the visit τ_i with the maximum waiting time $w_i^{\text{earliest}} = \max\left\{w_k^{\text{earliest}}, \dots, w_{l-1}^{\text{earliest}}\right\}$ where ties are broken randomly. The rationale behind this idea is that large w_h^{earliest} values often indicate the visits in an optimal solution, where extra waiting time actually is introduced to satisfy the separation-time constraints.

Utilizing waiting times computed by earliest possible arrival times works well for the majority of instances but for some instances the ζ_h values are altered unfavorably so that the instances become infeasible. To counteract this problem, we propose alternative waiting times which are calculated using arrival times with minimum tour duration: $w_h^{\text{ms}} = \max\left\{0, a_h^{\text{ms}} - a_{h-1}^{\text{ms}} - \zeta_{h-1}\right\}$, $\forall h = 1, \dots, l$. Visits with waiting times larger than 0 indicate visits in the tour with minimum tour duration for which additional waiting time had to be introduced in order to satisfy separation-time constraints. Using w_h^{ms} waiting times we can effectively complement situations where the approach utilizing w_h^{earliest} values yields infeasible or low-quality solutions. Therefore, we solve the DP-based heuristic twice, using both w_h^{earliest} and w_h^{ms} and take the best solution.

Even if the solution of this DP-based heuristic does not guarantee optimality in general, it works well in practice, producing near optimal solutions in significantly shorter computation times than the exact LP approach.

6 Large Neighborhood Search for the DRPSC-STW

Our overall approach for solving the DRPSC-STW follows the classical large neighborhood search metaheuristic [6] with an embedded variable neighborhood descent (VND) for local improvement.

We define our *destroy* and *repair* methods as follows. In order to destroy a current solution candidate, we select two out of m districts uniformly at random and remove all objects from these districts. The removed objects are copied to a so called *ejection pool*. Then, we apply the repair phase of the *district elimination algorithm* proposed by Prischink et al. [8]. The algorithm continues until all objects in the ejection pool are reassigned to the available districts. Using this *destroy* and *repair* methods, we guarantee that the solution stays feasible with the same number of districts. At each LNS iteration a VND is applied to locally improve the incumbent solution.

6.1 Variable Neighborhood Descent

We use three common neighborhood structures from the literature and search in a best improvement fashion. We apply these neighborhoods in the given order since we could not identify any significant advantages using different orderings. Infeasible solutions are discarded.

2-opt: Classical 2-opt neighborhood where all edge exchanges are considered.
swap: Exchanges all pairs of distinct visits within a route.
or-opt: Moves sequences of one to three consecutive visits at another place in the route.

The proposed VND is performed separately for each route of every district. Our local improvement component could also be very well parallelized since different routes can be optimized independently of each other, however this is not in the scope of this work. Since routes having no penalties are already optimal, they are excluded from local improvement.

7 Computational Results

For the computational results, we used the instances which have been created by Prischink et al. [8]. In a first optimization round, we solve the districting part of the DRPSC-STW by means of the district elimination algorithm proposed by Prischink et al., based on the hard time windows only, generating input[3] for the subsequent time window penalty minimization round with the LNS algorithm. As global parameters we have chosen $t^{\max}$ to be 12 hours and the maximum allowed penalty $\Delta = 60$ minutes, which represent typical values used in practical settings. Furthermore, we set HH (Algorithm 1) specific parameters $\delta = 1$ and $\varepsilon = 30$, which have been determined empirically. For our test instances, they give good balance between computational speed and accuracy. Every instance was given a maximum allowed time limit of 900 seconds for the execution of the LNS and we have performed 20 runs for every instance. All tests have been executed as single threads on an Intel Xeon E5540 2.53GHz Quad Core processor. The algorithms have been written in C++ and have been compiled with gcc-4.8 and for solving the LP we used Gurobi 7.0.

In Table 1 the results of the LNS-LP and LNS-HH can be found. In the instance column, we specify the instance parameters. Sequentially, the name of the used TSPlib instance (refer to Prischink et al. [8] for a more detailed description), the number of runs performed, the number of objects $|I|$, the total number

[3] `https://www.ac.tuwien.ac.at/files/resources/instances/drpsc/evoc17.tgz`

Table 1. Results of the LNS with embedded LP and HH as solution evaluation function

Instance							**LNS-LP**				**LNS-HH**			
name	runs	$\|I\|$	$\|V\|$	α	β	v	#best	$\overline{\text{obj}}$ [s]	$\bar{t}$ [s]	#eval	#best	$\overline{\text{obj}}$ [s]	$\bar{t}$ [s]	#eval
berlin52_1	20	51	133	0.0	0.7	4	13	476.9	640.2	499,624.3	**20**	29.9	298.0	757,181.5
berlin52_2	20	51	130	0.0	0.7	4	13	235.1	662.5	525,862.1	**19**	7.9	239.0	586,096.8
berlin52_3	20	51	140	0.0	0.7	4	5	3,230.9	900.0	595,039.3	**15**	1,169.7	900.0	938,946.7
ch150_1	20	149	360	0.2	0.5	4	4	88,910.1	900.0	663,320.8	**16**	55,234.8	900.0	1,308,850.8
ch150_2	20	149	402	0.2	0.5	4	0	164,399.8	900.0	662,869.3	**20**	80,989.3	900.0	1,257,789.1
ch150_3	20	149	357	0.2	0.5	4	3	78,979.9	900.0	748,549.8	**17**	36,370.5	900.0	1,620,281.2
ft70_1	20	69	167	0.1	0.5	4	2	5,035.8	900.0	993,374.5	**18**	1,196.9	900.0	2,815,488.6
ft70_2	20	69	180	0.1	0.5	4	1	3,087.0	900.0	975,529.6	**20**	464.2	878.3	2,719,290.8
ft70_3	20	69	144	0.1	0.5	4	3	5,602.2	900.0	918,004.6	**17**	1,509.0	900.0	2,496,826.1
gr48_1	20	47	120	0.2	0.7	4	8	92,669.4	900.0	284,934.1	**12**	70,082.1	900.0	443,507.7
gr48_2	20	47	115	0.2	0.7	4	3	9,445.5	900.0	851,306.0	**17**	4,233.1	900.0	2,296,485.5
gr48_3	20	47	125	0.2	0.7	4	5	28,606.2	900.0	392,935.5	**15**	24,982.1	900.0	615,803.7
rd100_1	20	99	152	0.1	0.5	2	3	25,824.7	900.0	876,710.0	**17**	11,050.4	900.0	2,289,110.3
rd100_2	20	99	160	0.1	0.5	2	4	22,367.5	900.0	828,483.6	**16**	7,618.6	900.0	2,123,393.7
rd100_3	20	99	152	0.1	0.5	2	4	12,132.5	900.0	826,517.7	**17**	2,136.9	900.0	2,036,959.2
st70_1	20	69	105	0.1	0.7	2	5	15,052.1	900.0	755,594.8	**16**	4,380.1	900.0	1,761,210.0
st70_2	20	69	91	0.1	0.7	2	4	18,622.9	900.0	806,985.2	**16**	8,228.9	900.0	2,126,834.2
st70_3	20	69	106	0.1	0.7	2	3	7,022.6	900.0	696,001.3	**20**	380.5	673.1	1,140,718.4
tsp225_1	20	224	334	0.2	0.7	2	0	272,118.2	900.0	969,287.0	**20**	183,974.1	900.0	1,904,494.3
tsp225_2	20	224	341	0.2	0.7	2	0	340,426.5	900.0	692,597.6	**20**	293,867.6	900.0	1,375,567.5
tsp225_3	20	224	332	0.2	0.7	2	0	161,586.6	900.0	710,581.4	**20**	141,153.5	900.0	1,471,799.2
						Average	4.0	64,563.4	884.5	727,338.5	**17.5**	44,240.9	828.0	1,623,173.1

of visits $|V|$, the percentage of large time windows (α), the percentage of mid-sized time windows (β) and the maximum number of allowed visits per object v is given. For the LNS-LP and LNS-HH the number of times the corresponding approach yields the best result, the average objective value over all runs of the instance, the average runtime, and the average number of objective function evaluations are given. Results show clearly that LNS-HH yields better objective values than LNS-LP since it is able to perform much more iterations within the given time limit due to fast objective function evaluations. It is also obvious that by increasing the instance size, the advantage of the efficient HH evaluation function is getting more pronounced. Moreover, a Wilcoxon signed-rank test shows that all observed differences on the overall number of best solutions among the LNS-LP and the LNS-HH are statistically significant with an error level of less than 1%.

We can conclude that LNS-HH is superior compared to LNS-LP due to significant performance advantage in the evaluation function, even though the HH-based evaluation function is only a heuristic method which in general does not yield proven optimal solutions although it can be observed that the optimality gap of HH is in most cases neglectably small.

8 Conclusions and Future Work

In this work we analyze the DRPSC-STW where the already introduced DRPSC is extended by soft time windows. This problem is of high practical relevance as it is possible to significantly improve solution quality by introducing only a negelectable penalty.

As metaheuristic we propose an LNS for approaching the DRPSC-STW. A critical bottleneck of our LNS is the evaluation of solution candidates where one has to find the minimum penalty given a particular visit order. We show that this evaluation function can be efficiently implemented by an LP-based approach, and furthermore we developed a sophisticated hybrid heuristic which was able to drastically outperform the LP-based variant.

We have formulated an efficient method to determine optimal arrival times of a given visit order which can be embedded inside a metaheuristic framework to solve the penalty minimization part of the DRPSC-STW. On the one hand this is not only relevant for the DRPSC-STW, as soft time windows play in general an important role in many practical scenarios.

Future research goals include the extension of the current LNS by incorporating adaptiveness into the destroy and repair moves. Furthermore, the authors want to note that it is also possible to extend the VND local search into a VNS by including a shaking neighborhood like randomized k-swap neighborhood, c.f. [1]. This way, one can combine micro- and macro-diversifications during the search.

References

1. Davidovic, T., Hansen, P., Mladenovic, N.: Variable neighborhood search for multiprocessor scheduling problem with communication delays. In: Proc. MIC. vol. 4, pp. 737–741 (2001)
2. Fagerholt, K.: Ship scheduling with soft time windows: An optimisation based approach. European Journal of Operational Research 131(3), 559–571 (2001)
3. Hashimoto, H., Ibaraki, T., Imahori, S., Yagiura, M.: The vehicle routing problem with flexible time windows and traveling times. Discrete Applied Mathematics 154(16), 2271–2290 (2006)
4. Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., Yagiura, M.: Effective local search algorithms for routing and scheduling problems with general time-window constraints. Transportation Science 39(2), 206–232 (2005)
5. Ioachim, I., Gelinas, S., Soumis, F., Desrosiers, J.: A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Networks 31(3), 193–204 (1998)
6. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, chap. 13, pp. 399–419. Springer (2010)
7. Prischink, M.: Metaheuristics for the Districting and Routing Problem for Security Control. Master's thesis, TU Wien, Institute of Computer Graphics and Algorithms (May 2016), `https://www.ac.tuwien.ac.at/files/pub/prischink_16.pdf`, supervised by G. Raidl, B. Biesinger, and C. Kloimüllner
8. Prischink, M., Kloimüllner, C., Biesinger, B., Raidl, G.R.: Districting and routing for security control. In: Blesa, M.J., Blum, C., Cangelosi, A., Cutello, V., Nuovo, A.D., Pavone, M., Talbi, E.G. (eds.) Hybrid Metaheuristics: 10th International Workshop, HM 2016. Lecture Notes in Computer Science, vol. 9668, pp. 87–103. Springer (2016)
9. Taillard, É., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 31(2), 170–186 (1997)