Districting and Routing for Security Control*

Michael Prischink[1,2], Christian Kloimüllner[1], Benjamin Biesinger[1], and Günther R. Raidl[1]

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

Abstract. Regular security controls on a day by day basis are an essential and important mechanism to prevent theft and vandalism in business buildings. Typically, security workers patrol through a set of objects where each object requires a particular number of visits on all or some days within a given planning horizon, and each of these visits has to be performed in a specific time window. An important goal of the security company is to partition all objects into a minimum number of disjoint clusters such that for each cluster and each day of the planning horizon a feasible route for performing all the requested visits exists. Each route is limited by a maximum working time, must satisfy the visits' time window constraints, and any two visits of one object must be separated by a minimum time difference. We call this problem the *Districting and Routing Problem for Security Control*. In our heuristic approach we split the problem into a districting part where objects have to be assigned to districts and a routing part where feasible routes for each combination of district and period have to be found. These parts cannot be solved independently though. We propose an exact mixed integer linear programming model and a *routing construction heuristic* in a greedy like fashion with *variable neighborhood descent* for the routing part as well as a *districting construction heuristic* and an *iterative destroy & recreate* algorithm for the districting part. Computational results show that the exact algorithm is only able to solve small routing instances and the iterative destroy & recreate algorithm is able to reduce the number of districts significantly from the starting solutions.

1 Introduction

As in the area of private security control constant surveillance of an object might not be economically viable or even necessary, security firms face the problem of

* We want to thank Günter Kiechle and Fritz Payr from CAPLAS GmbH for the collaboration on this topic. This work is supported by the Austrian Science Fund (FWF) grant P24660-N23 and by the Austrian Research Promotion Agency (FFG) under contract 849028.

sending security guards to visit a large number of sites multiple times over the course of a day in order to fulfill their custodial duty.

Security companies have to schedule tours for their employees in order to cover all needed visits of all objects under their guardianship. The complexity of this task leaves a high potential for solving this problem by algorithmic techniques to minimize the number of needed tours. Thus, we propose the *Districting and Routing Problem for Security Control* (DRPSC) which consists of a districting part and a routing part. In the districting part all objects should be partitioned into a minimum number of disjoint districts, such that a single district can be serviced by a single security guard within each working day of a planning horizon. Given such a partitioning a routing problem has to be solved for each combination of district and day. We seek for a tour starting and ending at a central location which satisfies a maximum tour duration and the time window constraints for each requested visit. In case multiple visits are required at an object in the same period, there typically has to be a separation time between consecutive visits to ensure a better distribution over time. To minimize the number of districts, it is important to minimize the duration of the planned tours in order to incorporate as many objects into the resulting districts as possible, which shows the inseparability of the districting and routing parts.

We address the routing part by an exact *mixed integer linear programming formulation* (MIP) and a *routing construction heuristic* (RCH) with a subsequent *variable neighborhood descent* (VND). For the districting part we propose an *iterative destroy & recreate* (IDR) approach based on an initial solution identified by a *districting construction heuristic* (DCH).

This article is structured as follows. In Section 2 a formal problem definition of the DRPSC is given followed by a survey of related work in the literature in Section 3. The proposed algorithms for solving the routing subproblem and the districting problem are described in Sections 4 and 5, respectively. Computational results are shown and discussed in Section 6, before final conclusions are drawn and an outlook for possible future work is given in Section 7.

2 Problem Definition

This section formalizes the DRPSC. We are given a set of objects $I = \{1, \ldots, n\}$ and a starting location, which we call in relation to the usual terminology in vehicle routing depot 0. There are p planning periods (days) $P = \{1, \ldots, p\}$, and for each object $i \in I$ a set of visits $S_i = \{i_1, \ldots, i_{|S_i|}\}$ is defined. Not all visits, however, have to take place in each period. The visits requested in period $j \in P$ for object $i \in I$ are given by subset $W_{i,j} \subseteq S_i$.

For each visit $i_k \in S_i$, $i \in I$, $k = 1, \ldots, |S_i|$, we are given its duration $t^{\text{visit}}_{i_k} \geq 0$ and a time window $T_{i_k} = [T^{\text{e}}_{i_k}, T^{\text{l}}_{i_k}]$, during which the whole visit has to take place. The time windows of successive visits of an object may also overlap but visit i_k always has to take place before a visit $i_{k'}$ with $k, k' \in W_{i,j},\ k < k'$ and they must be separated by a minimum duration of t^{sep}. The maximum duration of each planned tour must not exceed a global maximum duration $t^{\max}$.

Next, we define underlying graphs on which our proposed algorithms operate. For each period $j \in P$ we define a directed graph $G^j = (V^j, A^j)$ where V^j refers to the set of visits requested at corresponding objects, i.e., $V^j = \bigcup_{i \in I} W_{i,j}$, and the arc set is: $A^j = \{(i_k, i'_{k'}) \mid i_k \in W_{i,j}, i'_{k'} \in W_{i',j}\} \setminus \{(i_k, i_{k'}) \mid i_k, i_{k'} \in W_{i,j}, k' \leq k\}$. We have arc weights associated with every arc in A^j which are given by $t^{\text{travel}}_{i,i'}$, the duration of the fastest connection from object i to object i'. We assume that the triangle inequality holds among these travel times. Let us further define the special nodes 0_0 and 0_1 representing the start and end of a tour and the augmented node set $\hat{V}^j = V^j \cup \{0_0, 0_1\}$, $\forall j \in P$. Accordingly, we add outgoing arcs from node 0_0 to all visits $i_k \in V^j$ and arcs from all visits $i_k \in V^j$ to node 0_1, formally, $\hat{A}^j = A^j \cup \{(0_0, i_k) \mid i_k \in V^j\} \cup \{(i_k, 0_1) \mid i_k \in V^j\}$. Consequently, we define the augmented graph $\hat{G}^j = (\hat{V}^j, \hat{A}^j)$.

The goal of the *DRPSC* is to assign all objects in I to a smallest possible set of districts $R = \{1, \ldots, \delta\}$, i.e., to partition I into δ disjoint subsets I_r, $r \in R$, with $I_r \cap I_{r'} = \emptyset$ for $r, r' \in R,\ r \neq r'$ and $\bigcup_{r \in R} I_r = I$, so that a feasible tour $\tau_{r,j}$ exists for each district I_r, $r \in R$ and each planning period $j \in P$. A tour $\tau_{r,j} = (\tau_{r,j,0}, \tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}}, \tau_{r,j,l_{r,j}+1})$ with $\tau_{r,j,0} = 0_0, \tau_{r,j,l_{r,j}+1} = 0_1$, $l_{r,j} = \sum_{i \in I_r} |W_{i,j}|$, and $\tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}} \in \bigcup_{i \in I_r} W_{i,j}$ has to start at the depot node 0_0, has to perform each visit $i_k \in W_{i,j}$ in the respective sequence for each object $i \in I_r$ exactly once, and finally has to return back to the depot, i.e., reach node (0_1). A tour $\tau_{r,j}$ is feasible if each visit $\tau_{r,j,u}$, $u = 0, \ldots, l_{r,j} + 1$ takes place in its time window T_{i_k}, where waiting before a visit is allowed, the minimum duration t^{sep} between visits of the same object is fulfilled, and the total tour duration does not exceed $t^{\max}$.

Note that the routing part can be solved for a given district I_r and each period $j \in P$ independently and consists of finding a feasible tour $\tau_{r,j}$.

3 Related Work

To the best of our knowledge there is no work covering all the aspects of the DRPSC as considered here. The similarity of the DRPSC to the vehicle routing problem with time windows (VRPTW), however, leads to a huge amount of related work. A majority of the approaches in the literature aim at minimizing the total route length without taking the makespan into account [4, 6, 9, 11–13, 15, 16]. As the practical difficulty usually increases when makespan minimization is considered, specialized algorithms have been developed for the traveling salesman problem with time windows (TSPTW) [3, 5], which is the specialization of the VRPTW to just one tour. The routing part of the DRPSC is similar to the TSPTW as the aim is to find a feasible tour of duration less than a prespecified value which is related to the minimization problem of the TSPTW. In the TSPTW, however, multiple visits of the same objects and a separation time between them are not considered. Interestingly, López-Ibáñez et al. [7] showed that by adapting two state-of-the-art metaheuristics for travel time minimization of the TSPTW [6, 12] to makespan minimization it is possible to outperform the specialized algorithms. Many of the proposed approaches focus on first minimiz-

ing the number of needed routes and only in a second step minimizing the travel time or makespan, e.g., by using a hierarchical objective function [11, 13]. Nagata and Bräysy [10] propose a route minimization heuristic which in particular tries to minimize the number of routes needed to solve the VRPTW. They also rely on a destroy-and-recreate heuristic which iteratively tries to delete one route while maintaining an ejection pool (EP). The EP stores all objects which are yet to be inserted. The algorithm tries to identify objects which are difficult to insert in one of the current routes and utilizes this information for choosing objects to be removed and re-inserted. As this approach produced excellent results we adopt this basic idea of the destroy-and-recreate heuristic here.

Exact solution approaches for the VRPTW were proposed by Ascheuer et al. [1] who developed a branch-and-cut algorithm using several valid inequalities and were able to solve most instances with up to 50–70 nodes. Baldacci et al. [2] introduce the ngL-tour relaxation. By using column generation as well as dynamic programming they are able to solve instances with up to 233 nodes to optimality and report new optimal solutions that have not been found previously. A current state-of-the-art method for heuristically solving several variants of the VRPTW is a hybrid genetic algorithm (GA) by Vidal et al. [16]. As many other approaches described in the literature [11, 13] they use a penalty function for handling infeasible routes, which is described in [11]. In the GA the initial solutions are created randomly but there are also more elaborate construction heuristics available: Solomon [15] proposes several algorithms for constructing only feasible solutions by extending the well-known savings heuristic, a nearest neighbor heuristic, and insertion heuristics using different criteria. Numerous simple construction heuristics for the asymmetric TSPTW are also proposed by Ascheuer et al. [1].

4 Routing Problem

An important factor when approaching the DRPSC is a practically efficient approach to the underlying routing problems. This part is embedded in the whole approach for optimizing the districting as a subcomponent which is called when the feasibility of a district needs to be checked. As already mentioned, this subproblem is similar to the well-known TSPTW which has been exhaustively studied in the literature. There is, however, one substantial and significant difference: objects have to be visited several times per period and between every two visits of the same object there has to be a specific separation time. Nevertheless, many fruitful ideas of the literature can be adopted to our problem.

As a single routing problem is solved for each period $j \in P$ and each district $r \in R$ independently, we are given one graph $G_r^j = (V_r^j, A_r^j)$. The node set is defined as $V_r^j = V^j \cap \bigcup_{i \in I_r} W_{i,j}$ and the arc set as $A_r^j = A^j \setminus \{(i_k, i'_{k'}) \mid i_k \notin V_r^j \vee i'_{k'} \notin V_r^j\}$. Similarly, we define the augmented graph containing the tours' start and end nodes 0_0 and 0_1 as $\hat{G}_r^j = (\hat{V}_r^j, \hat{A}_r^j)$ where $\hat{V}_r^j = \hat{V}^j \cap \bigcup_{i \in I_r} W_{i,j}$ and $\hat{A}_r^j = \hat{A}^j \setminus \{(i_k, i'_{k'}) \mid i_k \notin \hat{V}_r^j \vee i'_{k'} \notin \hat{V}_r^j\}$.

For computing the duration of a tour τ we first define the arrival and waiting times for each visit of the tour. Moreover, let us define the auxiliary function $\kappa : V_r^j \mapsto I$ which maps the visit $i_k \in V_r^j$, to its corresponding object $i \in I_r$, and the auxiliary function $\gamma : V_r^j \mapsto \mathbb{N}$ which maps visit $i_k \in V_r^j$, to its corresponding index in the set of visits for this particular object. For every visit $i_k \in V_r^j$, a_{i_k} denotes the arrival time at the object, whereas a_{0_0} and a_{0_1} denote the departure and arrival time for the depot nodes 0_0 and 0_1, respectively. Let $t^{\text{wait}}_{\tau_u} = \max(0, T^e_{\tau_u} - \max(a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}))$ denote the waiting time before a visit τ_u can be fulfilled. We aim at finding a feasible tour $\tau = (0_0, \tau_1, \ldots, \tau_l, 0_1)$, $\tau_1, \ldots, \tau_l \in V_r^j, l = |V_r^j|$ through all visits starting and ending at the depot such that the total tour duration $T(\tau) = a_{0_1} - a_{0_0}$ does not exceed $t^{\max}$.

4.1 Exact Mixed Integer Linear Programming Model

The following compact mixed integer programming (MIP) model operates on the previously defined and reduced graph G_r^j and is based on Miller-Tucker-Zemlin (MTZ) [8] constraints. We use binary decision variables $y_{i_k,i'_{k'}}$ $\forall (i_k, i'_{k'}) \in A_r^j$ which are set to 1 if the arc between the k-th visit of object i and the k'-th visit of object i' is used in the solution, and 0 otherwise. We model arrival times by additional continuous variables a_{i_k} $\forall i_k \in V_r^j$ and ensure by these variables compliance with the time windows and the elimination of subtours. For each district $r \in R$ and each period $j \in P$ we solve the following model:

$$\min \quad \sum_{i_k \in V_r^j} (t^{\text{wait}}_{i_k} + t^{\text{visit}}_{i_k}) + \sum_{(i_k, i'_{k'}) \in \hat{A}_r^j} (y_{i_k,i'_{k'}} \cdot t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \tag{1}$$

$$\text{s.t.} \quad \sum_{(i_k, i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = \sum_{(i'_{k'}, i_k) \in \hat{A}_r^j} y_{i'_{k'},i_k} \qquad \forall i_k \in V_r^j \tag{2}$$

$$\sum_{(0_0, i_k) \in \hat{A}_r^j} y_{0_0,i_k} = 1 \tag{3}$$

$$\sum_{(i_k, 0_1) \in \hat{A}_r^j} y_{i_k,0_1} = 1 \tag{4}$$

$$a_{i_k} - a_{i'_{k'}} + t^{\max} \cdot (1 - y_{i'_{k'},i_k}) \geq t^{\text{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\text{visit}}_{i'_{k'}} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{5}$$

$$a_{i_k} + t^{\text{travel}}_{0,\kappa(i_k)} \cdot (1 - y_{0_0,i_k}) \geq t^{\text{travel}}_{0,\kappa(i_k)} \qquad \forall (0_0, i_k) \in \hat{A}_r^j \tag{6}$$

$$t^{\text{wait}}_{i_k} + t^{\max} \cdot (1 - y_{i_k,i'_{k'}}) \geq a_{i'_{k'}} - a_{i_k} - t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t^{\text{visit}}_{i_k} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{7}$$

$$a_{i_{k-1}} \leq a_{i_k} - t^{\text{sep}} \qquad \forall i_k, i_{k-1} \in V_r^j \tag{8}$$

$$\sum_{(i_k, i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = 1 \qquad \forall i_k \in V_r^j \tag{9}$$

$$T^e_{i_k} \leq a_{i_k} \leq T^l_{i_k} - t^{\text{visit}}_{i_k} \qquad \forall i_k \in V_r^j \tag{10}$$

$$y_{i_k,i'_{k'}} \in \{0, 1\} \qquad \forall (i_k, i'_{k'}) \in \hat{A}_r^j \tag{11}$$

The objective function (1) minimizes the total makespan within which all object visits take place by summing up all visit times, travel times, and waiting times.

Equalities (2) ensure that the number of ingoing arcs is equal to the number of outgoing arcs for each node $i_k \in V_r^j$. Equalities (3) and (4) ensure that there must be exactly one ingoing and outgoing arc for the depot in each period $j \in P$. Inequalities (5) are used to recursively compute the arrival times for every visit. If an edge $(i_k, i'_{k'})$ is not used, then the constraint is deactivated. These inequalities can be individually lifted by using $(T^{\mathrm{l}}_{i'_{k'}} - t^{\mathrm{visit}}_{i'_{k'}}) + (t^{\mathrm{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\mathrm{visit}}_{i'_{k'}})$ instead of t^{tmax}, which is also done in our implementation. Inequalities (6) set the start time at the depot for each period. Inequalities (7) compute the waiting time at the k-th visit of object i before traveling to the k'-th visit of object i'. We need these waiting times $t^{\mathrm{wait}}_{i_k}$ $\forall i_k \in V_r^j$ for the objective function to minimize the makespan of the route. These inequalities can also be lifted by replacing $t^{\max}$ with the term $(T^l_{i'_{k'}} - t^{\mathrm{visit}}_{i'_{k'}}) - T^l_{i_k} - t^{\mathrm{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t^{\mathrm{visit}}_{i_k}$. Inequalities (8) model the minimum time required between two different visits of the same object, i.e., ensure the separation time t^{sep}. Inequalities (9) state that there must exist an ingoing and an outgoing arc for the k-th visit of object i, if this particular visit is requested in the considered period $j \in P$. It is ensured that every time window of every visit $i_k \in V_r^j$ is fulfilled in (10). In (11) the domain definitions for the binary edge-decision variables $y_{i_k,i'_{k'}}$ are given.

In the context of the districting problem we use this model only for checking feasibility which can usually be done faster than solving the optimization problem to optimality. To this end we replace the objective function by $\min\{0\}$ and add the following constraints for limiting the makespan to $t^{\max}$:

$$\sum_{i_k \in V_r^j} (t^{\mathrm{wait}}_{i_k} + t^{\mathrm{visit}}_{i_k}) + \sum_{(i_k, i'_{k'}) \in \hat{A}_r^j} (y_{i_k,i'_{k'}} \cdot t^{\mathrm{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \leq t^{\max} \tag{12}$$

4.2 Heuristics

For larger districts the exact feasibility check using the MIP model might be too slow, hence we also propose a faster greedy construction heuristic followed by a variable neighborhood descent.

Given a sequence of visits τ, we first determine if a tour can be scheduled such that the time window constraints of all visits are satisfied. For this purpose, we compute the earliest possible arrival time a_{i_k} for each visit and minimize waiting times.

Feasibility of a tour: Since the (intermediate) tour τ starts at the depot at the earliest possible time, the departure at the depot a_{0_0} is set to 0. For each subsequent visit τ_u, the arrival time a_{τ_u} is the maximum of $T^e_{\tau_u}$ and the arrival time at the preceding visit $a_{\tau_{u-1}}$ including visit time $t^{\mathrm{visit}}_{\tau_{u-1}}$ and travel time $t^{\mathrm{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}$ from the preceding visit's object $\kappa(\tau_{u-1})$ to the current visit's object $\kappa(\tau_u)$. The depot has no requested visit times, therefore we define $t^{\mathrm{visit}}_{0_0} = t^{\mathrm{visit}}_{0_1} = 0$. Furthermore, for each object i the separation time t^{sep} between visit i_k and i_{k-1} for all $k > 1$ has to be respected. Formally:

$$a_{0_0} = 0$$

$$a_{\tau_u} = \begin{cases} \max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}\} \\ \qquad\qquad \text{for } u > 1, \gamma(\tau_u) = 1 \\ \max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} \\ \qquad\qquad \text{for } u > 1, \gamma(\tau_u) > 1 \end{cases}$$

$$a_{0_1} = a_{\tau_l} + t^{\text{visit}}_{\tau_l} + t^{\text{travel}}_{\kappa(\tau_l),0}$$

If for any arrival time a_{i_k} with $i_k \in V_r^j$ the following condition is violated, the sequence of visits is infeasible:

$$a_{i_k} + t^{\text{visit}}_{i_k} \leq T^{\text{l}}_{i_k} \tag{13}$$

The resulting tour duration $T(\tau) = a_{0_1} - a_{0_0}$ can be minimized while keeping τ feasible by delaying the departure at the depot by the so called *forward time slack* proposed by Savelsbergh [14] for the TSPTW. The *forward time slack* $F(\tau_u, \tau_{u'})$ for the partial tour $\tau' = \tau_u, \ldots, \tau_{u'}$ adapted to our problem is

$$F'(\tau_u, \tau_{u'}) = \begin{cases} T^l_{\tau_u} - t^{\text{visit}}_{\tau_u} - a_{\tau_u} & \text{for } u = u' \\ F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})} & \text{for } u' > 1, \gamma(\tau_{u'}) = 1 \\ \min\{F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})}, \\ \qquad F'(\tau_u, \tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}) - T^l_{\tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}} + T^l_{u'} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{sep}}\} \\ \qquad\qquad \text{for } u' > 1, \gamma(\tau_{u'}) > 1 \end{cases}$$

$$F(\tau_u, \tau_{u'}) = \min_{v=u,\ldots,u'} \{F'(\tau_u, \tau_v)\} \tag{14}$$

The tour duration of tour τ must not exceed $t^{\max}$. Formally, if

$$T(\tau) - \min(F(0_0, 0_1), \sum_{u=1}^{l} t^{\text{wait}}_{\tau_u}) < t^{\max} \tag{15}$$

holds, the sequence τ is feasible, otherwise infeasible.

Routing Construction Heuristic: We developed a *Routing Construction Heuristic* (RCH) based on an insertion heuristic by starting from a partial tour $\tau' = (0_0, 0_1)$ containing only the start and end nodes and iteratively adding all visits $i_k \in V_r^j$ to τ'. A 2-step approach is used, where we first order the visits according to some criteria and then insert them at the first feasible or best possible insert position, respectively. For the insertion order we compute the *flexibility value* of each visit $i_k \in V_r^j$ where visits with less flexibility are inserted first:

$$\mathit{flex}(i_k) = T^l_{i_k} - T^e_{i_k} - t^{\text{visit}}_{i_k} \tag{16}$$

$$\mathit{flex}(i_k^{(1)}) \leq \mathit{flex}(i_k^{(2)}) \leq \cdots \leq \mathit{flex}(i_k^{(|V_r^j|)}) \tag{17}$$

Visits with less flexibility may be more difficult to insert as they need to be scheduled at a very specific time. Ties are broken randomly.

In a second phase we start by trying to insert the first visit, i.e., $i_k^{(1)}$, into the partial tour τ'. We start at the front, i.e., try to insert it after the start node 0_0, and move backwards to the end. Then, we either stop when we found the first feasible insert position in the first feasible variant or we compute insertion costs for each possible insert position and insert the visit at the position with the minimum costs for the best possible insertion variant. We define these costs as:

$$d_{i_k,u'} = \begin{cases} a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} - a_{\tau_u} & \text{if (19) and (20) hold} \\ \infty & \text{otherwise} \end{cases} \tag{18}$$

These insertion costs $d_{i_k,u'}$ determine the amount of time by which the visit τ_u has to be moved backwards in order to insert the new visit $\tau_{u'}$. Note that $d_{i_k,u'}$ may also be negative, if the space for insertion of visit $\tau_{u'}$ is bigger than necessary. However, this is desirable as we use those insert positions more likely which have bigger gaps and smaller gaps are kept for later inserts. In Section 6 we compare both, the first feasible and the best possible variant, to each other in terms of solution quality and runtime.

We further maintain global variables for the *forward time slack* $F(\tau')$ and all arrival times a_{τ_u} of each partial tour τ' computed during the execution of the insertion heuristic. For an insertion to be feasible, the latest allowed arrival time at visit $\tau_{u'}$ must be greater or equal to the earliest possible arrival at that visit considering the previous visit's earliest arrival, its visit time and the travel time between τ_{u-1} and $\tau_{u'}$. Furthermore, the earliest departure at $\tau_{u'}$ including the travel time between $\tau_{u'}$ and τ_u must be smaller or equal to the earliest arrival at τ_u delayed by the *forward time slack* of the partial tour from τ_u to the depot. Using the definition of the forward time slack in equality (14) and if inequality (13) holds, then the insertion is feasible if, in addition, also the following two inequalities hold:

$$T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} \geq \max\{a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_{u'})}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} \tag{19}$$

$$a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} \leq a_{\tau_u} + F(\tau_u, 0_1) \tag{20}$$

Local improvement: If the solution found by the RCH is infeasible we additionally employ a VND to reduce the number of infeasibilities and possibly come to a feasible solution. First, we insert each infeasible visit i_k into the tour on the position u' where the costs $d_{i_k,u'}$ are minimum. We use a lexicographical penalty function to penalize infeasible tours where the first criterion is the number of time window violations and the second criterion is the duration of the route as proposed by López-Ibáñez et al. [6]. We use three common neighborhood structures from the literature and search them in a best improvement fashion in random order while respecting the visit order:

Swap: This neighborhood considers all exchanges between two distinct visits.

Algorithm 1 Districting Construction Heuristic

```
1: init: R ← {1}, U ← sort(I)
2: for all i ∈ U do
3:     inserted ← false
4:     for all r ∈ R do
5:         if insert(i, r) then
6:             inserted ← true
7:             break
8:         end if
9:     end for
10:    if not inserted then
11:        r' ← create |R| + 1-th new empty district
12:        R ← R ∪ {r'}
13:        insert(i, r')
14:    end if
15: end for
```

2-opt: This is the classical 2-opt neighborhood for the traveling salesman problem where all edge exchanges are checked for improvement.

Or-opt: This neighborhood considers all solutions in which sequences of up to three consecutive visits are moved to another place in the same route.

If at some point during the algorithm the value of the penalty function is zero we terminate with a feasible solution.

5 Districting Problem

In the previous section we have already introduced a fast heuristic for efficiently testing feasibility of a given set of objects by building a single tour for each period through all requested visits of these objects. In the districting part of the DRPSC we face the problem of intelligently assigning objects to districts such that the number of districts is minimized. For checking the feasibility of this assignment we use the previously introduced RCH. Alternatively, we could also use our MIP model for solving these subproblems but, as we will see in Section 6, it is too slow to be used in practical scenarios. We propose a DCH and an iterative destroy & recreate algorithm where the former generates an initial solution and the latter tries to iteratively remove districts.

5.1 Districting Construction Heuristic

Starting with one district, objects are iteratively added to the existing districts $r \in R$. Whenever adding an object $i \in U$ to any of the available districts $r \in R$ would make the assignment infeasible, $i \in U$ is added to a newly created district r'. The overall DCH is shown in Algorithm 1 and explained below.

First, the set of districts R is initialized with the first empty district 1 and the set of objects I is sorted by extending the flexibility values as defined in equation (16) from visits to objects. All objects are sorted by the sum of their flexibility values $\sum_{j \in P} \sum_{i_k \in W_{i,j}} \mathit{flex}(i_k)$ in ascending order. As in the RCH, the resulting set U is denoted as the set of unscheduled visits. The DCH terminates when all $i \in U$ have been scheduled (2) and, as a consequence, all requested

visits have been inserted successfully and we obtain a feasible solution to the DRPSC. The insertion of object i into district r (lines 5 and 13) is accomplished by checking for each scheduled visit $i_k \in W_{i,j}$ if i_k can be feasibly inserted into the particular district r (for the definition of feasibility of a tour see also Section 4.2). In line 5 the DCH inserts i either into the first feasible or into the best possible insert position, as described in Section 4.2. The *insert* function returns *false*, if no feasible insertion position is found for at least one $i_k \in W_{i,j}$, $\forall j \in P$. It returns *true*, if a feasible insertion position is found for each visit $i_k \in W_{i,j}$, $\forall j \in P$. If the loop over all districts (line 4) terminates without finding any feasible insertion position the variable *inserted* stays *false* and a new empty district is created in line 11. The proposed constructive algorithm will terminate with a feasible solution after $|U|$ iterations.

5.2 Iterative Destroy & Recreate

Nagata and Bräysy [10] proposed a *route elimination algorithm* for reducing the number of vehicles needed in the VRPTW. We apply the basic idea to the districting problem. The algorithm starts with the initial assignment where every object is reached by a separate route. Then, one district $r \in R$ is chosen for elimination at a time, maintaining all now unassigned objects in an *ejection pool* (EP). Then, it is tried to assign all objects of the EP to the remaining districts $R \setminus \{r\}$. If this is successful, the number of districts could be reduced by one and another district is chosen for elimination. We adapt this idea to the DRPSC and use the result of the DCH described in Section 5.1 as initial solution.

Let the assignment of an object $i \in I$ to a district $r \in R$ be feasible if and only if a feasible tour can be scheduled for all assigned visits of all objects for each period. Let c_i be a penalty value of object $i \in I$ denoting failed attempts of inserting object i into a district. Each time a visit cannot be inserted, this penalty value is increased by one, revealing objects which are difficult to assign to one of the available districts.

If the EP becomes empty, a feasible assignment of objects to districts is found. Subsequently, another iteration is started, destroying a district and reassigning its objects to the remaining ones. The overall district elimination algorithm is shown in Algorithm 2.

First, the EP is initialized to the empty set and the penalty values of all objects are set to 0. Starting with the solution provided by DCH a district is chosen for elimination in line 2. One of the following strategies is applied uniformly at random for selecting a district for elimination:

Minimum number of scheduled visits: This implies that only a minimum number of visits has to be reinserted to regain a feasible solution.

Shortest tour duration: Selecting a district where the maximum tour duration over all periods is minimal can be promising because this district might lead to a district with visits of shorter durations resulting in easier insert operations.

Maximum waiting times: Selecting a district with a loose schedule may indicate less or shorter visits, making them easier to reinsert.

Algorithm 2 District elimination algorithm

1: **init:** $EP \leftarrow \emptyset,\ c_i \leftarrow 0\ \forall i \in I$
2: choose a district $r^{\text{del}} \in R$ for deletion
3: $R \leftarrow R \setminus \{r^{\text{del}}\}$
4: $EP \leftarrow EP \cup \{i \mid i \in I_{r^{\text{del}}}\}$
5: **while** $EP \neq \emptyset \wedge$ termination criterion not met **do**
6: $\quad i^{\text{ins}} \leftarrow \arg\max_{i \in EP} \{c_i\}$
7: $\quad R_f \leftarrow$ feasible districts for assignment of i^{ins}
8: $\quad c_{i^{\text{ins}}} \leftarrow c_{i^{\text{ins}}} + |R| - |R_f|$
9: $\quad$ **if** $R_f \neq \emptyset$ **then**
10: $\quad\quad$ assign object i^{ins} to a randomly chosen feasible district $r \in R_f$
11: $\quad$ **else**
12: $\quad\quad$ select random district $r^{\text{ins}} \in R$
13: $\quad\quad$ assign object i^{ins} to district r^{ins}
14: $\quad\quad$ call VND for district r^{ins} (see Section 4.2)
15: $\quad\quad$ **while** $\exists$ an infeasible tour for any period of district r^{ins} **do**
16: $\quad\quad\quad i^{\text{del}} \leftarrow \arg\min_{i \in I_{r^{\text{ins}}}} \{c_i\}$
17: $\quad\quad\quad I_{r^{\text{ins}}} \leftarrow I_{r^{\text{ins}}} \setminus \{i^{\text{del}}\}$
18: $\quad\quad\quad EP \leftarrow EP \cup \{i^{\text{del}}\}$
19: $\quad\quad\quad$ call VND for district r^{ins} (see Section 4.2)
20: $\quad\quad$ **end while**
21: $\quad$ **end if**
22: **end while**

After deleting a district all objects of this district are moved to the EP (line 4). As long as the EP contains objects, we try to assign each object to one of the remaining districts. An object with maximum penalty value is chosen for the next assignment (6). For the chosen object i^{ins} the feasible districts for assignment are computed. If there is at least one feasible district for an assignment of object i^{ins} (9) we assign the object to such a district uniformly at random (10). If it is not possible to feasibly assign the object i^{ins} to any of the remaining districts we randomly choose a district for assignment (11). Then, we apply the VND described in Section 4.2 trying to make the district feasible. If this is not possible and the assignment is still infeasible we iteratively try to remove objects with lowest penalty values from this district r^{ins} in the following loop (15), remove them from district r^{ins} (17), and finally add them to the *EP* (18). Then again, we call the VND from Section 4.2 trying to make the resulting tour from the actual assignment feasible. After an iteration of the outer loop the object with highest penalty value of the *EP* has been inserted and other objects previously assigned to this district may have been added to the *EP*. The idea behind this approach is to insert difficult objects first and temporarily remove easy to insert objects from the solution to reinsert them later. When the EP is empty, a new best assignment with one district less is found. This algorithm iterates until a termination criterion, e.g., a time limit is met.

6 Computational Results

To evaluate our proposed algorithm computational tests are performed on a benchmark set of instances. As the DRPSC is a new problem we created new instances[3] based on the characteristics of real-world data provided by an indus-

[3] `https://www.ac.tuwien.ac.at/files/resources/instances/drpsc/hm16.tar.gz`

try partner. The main characteristics of real-world data are: Most of the time windows are of medium size, the depot is centralized among the objects, travel times are rather small with respect to visit times and the number of visits of the objects is usually ranged from 1 to 4. The distance matrix is taken from TSPlib instances and we added the depot, the visits and the time windows in the following way: The depot is selected by taking the node for which the total distance to all other nodes is a minimum. Each node of the original instance has between 1 and v visits, where v is a parameter of the instance. Small time windows have a length between 5 and 30 minutes, medium time windows have a length of 2, 3, 4, or 5 hours, and visits with large time windows are unrestricted. For the instance generation the length of a time window is assigned randomly to a visit based on parameter values α and β: a small time window is chosen with probability α, a medium time window with probability β, and a large time window with probability $1-\alpha-\beta$. Furthermore, we enforce that small and medium time windows of visits of the same object do not overlap and we choose the visit time uniformly at random from 3 to 20 minutes. For all our instances we set t^{sep} to 60 minutes and $t^{\max}$ to 10 hours.

The algorithm is implemented in C++ using Gurobi 6.5 for solving the MIP. For each combination of configuration and instance we performed 20 independent runs for the IDR while for the routing part we performed only one run because all tested algorithms for the routing part are deterministic. All runs were executed on a single core of an Intel Xeon processor with 2.54 GHz. The iterative destroy & recreate algorithm is terminated after a maximum of 900 CPU seconds. The MIP model for the routing part was aborted after 3600 CPU seconds.

In the first set of experiments the routing part of the DRPSC is examined more closely to evaluate RCH in comparison to the MIP model. Then, several configurations of our proposed algorithm for the whole problem are investigated.

6.1 Routing Part

First, the methods for the routing part are evaluated on a separate set of benchmark instances. In Table 1 the MIP model is compared to RCH, and RCH with the subsequent VND, denoted by RCH-VND. As the goal for the routing part is to minimize the makespan of a specific route, the maximum tour duration constraint is relaxed and the resulting makespan is given in minutes in the column *obj*. In the first four columns the instance parameters are specified. Sequentially, the instance name, the number of objects $|I|$, the maximum number of nodes of all objects $|V|$, the percentage of small (α), and medium time windows (β) and the maximum number of visits per objects v is given. For the RCH and RCH-VND we give the objective value (makespan in minutes) and the time needed for solving the instance. Then, the upper bound (UB), the lower bound (LB), the final optimality gap, and the time spent by Gurobi for solving the MIP model is shown. In the two remaining columns we present the relative gap between the MIP and RCH-VND $\Delta_{\text{MIP}} = (obj_{\text{RCH-VND}} - LB)/obj_{\text{RCH-VND}}$ as well as the relative gap between RCH and RCH-VND $\Delta_{\text{RCH}} = (obj_{\text{RCH}} - obj_{\text{RCH-VND}})/obj_{\text{RCH-VND}}$.

Table 1. Results of the MIP, RCH, and RCH-VND for the routing part.

Instance						RCH		RCH-VND		MIP				Rel. Difference	
name	$\|I\|$	$\|V\|$	α	β	v	*obj*	t[s]	*obj*	t[s]	UB	LB	Gap	t[s]	Δ_{MIP}	Δ_{RCH}
burma14_01	13	19	0	0.2	2	495.80	< 0.01	333.49	0.03	332.62	332.62	0.00%	2025.56	0.26%	48.67%
burma14_02	13	20	0	0.2	2	624.47	< 0.01	374.60	0.02	352.93	343.50	2.67%	3600.00	8.30%	66.70%
burma14_03	13	21	0	0.2	2	525.49	0.01	440.86	0.05	433.42	421.91	2.66%	3600.00	4.30%	19.20%
burma14_04	13	19	0	0.2	2	607.11	< 0.01	397.14	0.03	395.15	387.89	1.84%	3600.00	2.33%	52.87%
burma14_05	13	22	0	0.2	2	606.07	< 0.01	409.24	0.03	356.71	337.43	5.40%	3600.00	17.55%	48.10%
burma14_06	13	19	0	0.2	2	409.54	< 0.01	273.28	0.01	272.93	272.93	0.00%	2318.17	0.13%	49.86%
burma14_07	13	23	0	0.5	2	714.04	< 0.01	508.69	0.05	493.48	456.82	7.43%	3600.00	10.20%	40.37%
burma14_08	13	17	0	0.5	2	372.63	< 0.01	312.70	0.02	311.10	311.10	0.00%	2.09	0.51%	19.16%
burma14_09	13	21	0	0.5	2	416.55	< 0.01	370.61	0.04	360.71	360.71	0.00%	3.35	2.67%	12.40%
burma14_10	13	20	0	0.5	2	386.02	< 0.01	342.00	0.02	336.25	336.25	0.00%	10.74	1.68%	12.87%

In Table 1 we see that the MIP model is able to solve easier instances to optimality, but soon has very high running times. RCH-VND yields very reasonable solutions with objective values close to the LB of the MIP for most cases. When looking at the relative gap between the RCH and RCH-VND (Δ_{RCH}), we can conclude that the VND improves greatly on the objective value with only a minor increase in running time. Moreover, Δ_{MIP} reveals that RCH-VND produces results close to the results of the MIP, and for those instances where the relative gap between the MIP and RCH-VND is greater than 10%, the MIP also has a relatively larger gap between UB and LB.

As we require a fast method for deciding if a route is feasible within the districting problem, we conclude that RCH-VND is a reasonable choice.

6.2 Districting Part

For testing the proposed algorithms for the DRPSC we used three different configurations. In the IDR-DCH$^{\text{f}}$ algorithm we used the DCH for generating an initial feasible solution candidate with the first feasible strategy in contrast to the IDR-DCH$^{\text{b}}$ where we used a best possible strategy. Both configurations are compared with the IDR-SCH, where a simple construction heuristic (SCH) as proposed by Nagata and Bräysy [11] is used. In the SCH each object is put in a separate district which results in a trivial initial solution candidate.

In Table 2 the results of the experiments are shown. Columns *obj* show the average objective value, i.e., the minimum number of districts at the end of optimization, after the full run of IDR while columns obj_{f} and obj_{b} show the average objective value, i.e., the average number of districts, after the respective construction heuristic. Columns t^* show the median time in seconds after which the best solution has been found during the run of IDR while t_{f} and obj_{b} show the median time after which the respective construction heuristic has found an initial solution. Columns *sd* show the standard deviation of the objective value for 20 runs of a single instance.

We observe that for most instances the final objective value of the IDR is the same for all three configurations. There are, however, differences for the construction heuristics alone and DCH$^{\text{b}}$ for most but not all instances better

Table 2. Results of the DCH and IDR for the districting part.

Instance							IDR-SCH			IDR-DCH^{f}					IDR-DCH^{b}				
name	runs	$\|I\|$	$\|V\|$	α	β	v	*obj*	*sd*	t^*[s]	obj_f	t_f[s]	*obj*	*sd*	t^*[s]	obj_b	t_b	*obj*	*sd*	t^*[s]
st70_1	20	69	105	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	5	5	0.1	3.0	0.0	9
st70_2	20	69	91	0.1	0.7	2	3.0	0.0	4	6	< 0.1	3.0	0.0	6	5	0.1	3.0	0.0	6
st70_3	20	69	106	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	13	5	0.1	3.0	0.0	11
rd100_1	20	99	152	0.1	0.5	2	6.0	0.0	8	9	< 0.1	6.0	0.0	10	9	0.1	6.0	0.0	14
rd100_2	20	99	160	0.1	0.5	2	6.0	0.0	16	8	0.1	6.0	0.0	14	8	0.1	6.0	0.0	6
rd100_3	20	99	152	0.1	0.5	2	6.0	0.0	3	7	0.1	6.0	0.0	2	8	0.1	6.0	0.0	7
tsp225_1	20	224	334	0.2	0.7	2	11.0	0.0	47	13	0.2	11.0	0.0	64	13	0.4	11.0	0.0	121
tsp225_2	20	224	341	0.2	0.7	2	11.0	0.0	46	13	0.3	11.0	0.0	67	13	0.5	11.0	0.0	36
tsp225_3	20	224	332	0.2	0.7	2	10.0	0.0	226	12	0.3	10.0	0.0	257	12	0.5	10.0	0.0	177
gr48_1	20	47	120	0.2	0.7	4	5.0	0.0	6	8	< 0.1	5.0	0.0	4	7	< 0.1	5.0	0.0	14
gr48_2	20	47	115	0.2	0.7	4	5.0	0.0	12	7	< 0.1	5.0	0.0	7	7	< 0.1	5.0	0.0	10
gr48_3	20	47	125	0.2	0.7	4	4.0	0.0	527	7	< 0.1	4.0	0.0	438	6	< 0.1	4.0	0.0	488
berlin52_1	20	51	133	0.0	0.7	4	5.0	0.0	3	6	< 0.1	5.0	0.0	2	7	0.1	5.0	0.0	7
berlin52_2	20	51	130	0.0	0.7	4	5.0	0.0	5	7	< 0.1	4.0	0.0	142	6	0.1	5.0	0.0	1
berlin52_3	20	51	140	0.0	0.7	4	5.0	0.0	19	7	< 0.1	5.0	0.0	14	6	0.1	5.0	0.0	16
ft70_1	20	69	167	0.1	0.5	4	8.0	0.0	12	12	< 0.1	8.0	0.0	12	10	0.1	8.0	0.0	18
ft70_2	20	69	180	0.1	0.5	4	8.0	0.0	33	11	< 0.1	8.0	0.0	15	11	0.1	8.0	0.0	18
ft70_3	20	69	144	0.1	0.5	4	7.0	0.0	41	9	< 0.1	7.0	0.0	36	9	0.1	7.0	0.0	19
ch150_1	20	149	360	0.2	0.5	4	11.0	0.0	589	15	0.2	11.0	0.0	480	15	0.4	11.2	0.4	881
ch150_2	20	149	402	0.2	0.5	4	12.0	0.0	342	17	0.2	12.0	0.0	338	15	0.4	12.0	0.0	387
ch150_3	20	149	357	0.2	0.5	4	11.0	0.0	655	14	0.2	11.0	0.0	520	13	0.4	11.0	0.0	808

results but needed more time. The IDR-SCH works surprisingly well and was able to find good results in about the same amount of time as the other two (more sophisticated) configurations.

7 Conclusions and Future Work

In this work we introduced a new vehicle routing problem which originates from the security control sector. The goal of the Districting and Routing Problem for Security Control is to partition a set of objects under surveillance into disjoint clusters such that for each period a route through all requested visits can be scheduled satisfying complex time window constraints. As the objects may require multiple visits, there needs to be a minimum separation time between each two visits which imposes an interesting additional challenge. The proposed heuristic solution approach starts with a greedy construction heuristic followed by an iterate destroy and recreate algorithm. The latter works by iteratively destroying districts and trying to insert the resulting unassigned objects into the other districts. The computational results reveal that the MIP model is able to solve smaller instances of the routing problem to optimality and that the quality of the initial solutions of the districting problem has only a minor influence on the final solution quality. There are several possibilities for extending this algorithm in future work. As the feasibility check for a district is time-consuming a caching mechanism to prevent checking the same assignment of objects all over again seems promising. This could even be extended to checking subsets of such assignments, which also must be feasible if any superset of these objects

results in feasible routes. Another idea is to use neighborhood structures which exchange objects of two or more distinct clusters.

References

1. Ascheuer, N., Fischetti, M., Grötschel, M.: Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming 90(3), 475–506 (2001)
2. Baldacci, R., Mingozzi, A., Roberti, R.: New state-space relaxations for solving the traveling salesman problem with time windows. INFORMS Journal on Computing 24(3), 356–371 (2012)
3. Cheng, C.B., Mao, C.P.: A modified ant colony system for solving the travelling salesman problem with time windows. Mathematical and Computer Modelling 46(9), 1225–1235 (2007)
4. Da Silva, R.F., Urrutia, S.: A general VNS heuristic for the traveling salesman problem with time windows. Discrete Optimization 7(4), 203–211 (2010)
5. Gambardella, L.M., Taillard, E., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., Price, K.V. (eds.) New Ideas in Optimization, chap. 5, pp. 63–76. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999)
6. López-Ibáñez, M., Blum, C.: Beam-ACO for the travelling salesman problem with time windows. Computers & Operations Research 37(9), 1570–1583 (2010)
7. López-Ibáñez, M., Blum, C., Ohlmann, J.W., Thomas, B.W.: The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. Applied Soft Computing 13(9), 3806–3815 (2013)
8. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. Journal of the ACM 7(4), 326–329 (1960)
9. Mladenović, N., Todosijević, R., Urošević, D.: An efficient GVNS for solving traveling salesman problem with time windows. Electronic Notes in Discrete Mathematics 39, 83–90 (2012)
10. Nagata, Y., Bräysy, O.: A powerful route minimization heuristic for the vehicle routing problem with time windows. Operations Research Letters 37(5), 333–338 (2009)
11. Nagata, Y., Bräysy, O., Dullaert, W.: A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research 37(4), 724–737 (2010)
12. Ohlmann, J.W., Thomas, B.W.: A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS Journal on Computing 19(1), 80–90 (2007)
13. Prescott-Gagnon, E., Desaulniers, G., Rousseau, L.M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks 54(4), 190–204 (2009)
14. Savelsbergh, M.W.P.: The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Computing 4(2), 146–154 (1992)
15. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
16. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & Operations Research 40(1), 475–489 (2013)