Applying min-max *k* postmen problems to the routing of security guards[☆,☆☆]

Elias J. Willemse[*,a,1], Johan W. Joubert[b,2]

[a] *Logistics and Quantitative Methods, CSIR: Built Environment, PO Box 395, Pretoria, South Africa, 0001*
[b] *Industrial and Systems Engineering, University of Pretoria, South Africa, 0002*

Abstract

The most essential and alluring characteristic of a security estate is the estate's ability to provide 24-hour security to its residents, of which the continual patrolling of roads and paths is vital. The objective of this paper is to address the lack of sufficient patrol route design procedures by presenting a tabu search algorithm capable of generating multiple patrol routes for an estate's security guards. The paper shows that the problem of designing these routes can be modelled as an Arc Routing Problem, specifically as min-max *k* postmen problems. The algorithm is illustrated with a real problem instance from an estate in Gauteng, South Africa. The patrol routes generated by the algorithm provide a significant improvement in the even patrolling of the road network, and a more balanced work distribution among guards. The algorithm is also tested on several benchmark problems from literature.

Key words: Arc routing, Chinese postman problem, Rural postman problem, Tabu search algorithm, Security guard routing.

1. Introduction

Gated communities are a growing phenomenon in South Africa, reflecting an attempt by members of the public and developers to counteract the high levels of crime recorded within the country. Of the gated communities, security estates have become a popular choice for residence, mostly because of the estates' ability to provide 24-hour security to its inhabitants. Security systems of estates are designed using *crime prevention through environmental design* principles that rely upon the ability to influence offenders' decisions before they embark on criminal acts. The security system's main objective is not to identify and punish criminal activities, but to enhance the perceived risk of detection and apprehension. This approach requires highly visual security initiatives such as the patrolling of the estate's inner road and path network by security guards.

Designing patrol routes for security guards can become extremely complex as a result of the multitude of roads and paths that connect the estate's properties. The patrolling complexity is increased even further when certain essential conditions are taken into consideration: security guards cannot follow the same patrolling route every day as this predictable routing information could be used by unlawful parties to side-step the security guards. Moreover, all roads and paths have to be patrolled evenly as information regarding lesser patrolled roads and paths can be exploited. Patrolling of the roads and paths should also be evenly distributed among the guards to avoid discontentment and possible work overload.

A similar patrol route design problem, the overnight security service problem, is introduced by Wolfler Calvo and Cordone (2003). The problem deals with similar issues to security estate patrolling such as fair task assignment among the guards and unpredictable patrolling. However, the security service problem requires specific buildings and yards situated in a road network of a city to be inspected. Our application requires the complete road network of an estate to be patrolled, hence inspected.

[☆] This paper has been published in the *Journal of the Operational Research Society*, **63**(2), 245–260.
[☆☆] Last updated by: jwjoubert; Revision: 227 (2012-02-16 13:18:44 +0200 (Thu, 16 Feb 2012))
[*] Corresponding author
Email addresses: ewillemse@csir.co.za (Elias J. Willemse), johan.joubert@up.ac.za (Johan W. Joubert)
[1] Tel: +27 12 841 3934; Fax: +27 12 841 3037 (E.J. Willemse)
[2] Tel: +27 12 420 2843; Fax: +27 12 362 5103 (J.W. Joubert)

In this paper we describe the practical objectives and constraints of designing patrol routes, and show that the problem can be formulated as an Arc Routing Problem (ARP), more specifically as a min-max k-Rural Postmen Problem (MM k-RPP) or a min-max k-Chinese Postmen Problem (MM k-CPP). A tabu search algorithm capable of solving both these problems is proposed, and is illustrated with a real problem instance from an estate in Gauteng, South Africa. Results are compared with existing routes and schedules implemented in the estate. Our proposed solutions provide both a significant improvement in the even patrolling of the road network, and a more balanced work distribution among guards. The algorithm is also tested on benchmark problems found in literature. The solution quality for the real problem instance and the benchmark problems are assessed through lower bounds.

The remainder of this section presents a problem definition for designing patrol routes and discusses related work and solution approaches based on heuristic and metaheuristic strategies. Section 2 describes the proposed tabu search algorithm. In Section 3 we introduce three lower bounds and in Section 4 we illustrate the application of the tabu search algorithm on a real problem instance. Section 5 reports on computational results for the benchmark problems. Finally, we draw a few principal conclusions in Section 6 and provide directions for future work.

1.1. Problem definition and related work

Figure 1 shows a map of Midfield-Estate with roads and paths that can be traversed in any direction. The problem of designing patrol routes for such an estate can be formulated as an undirected Arc Routing

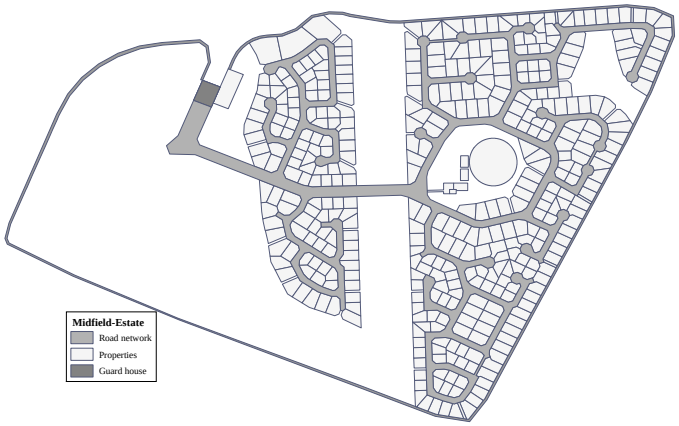

Figure 1: Midfield-Estate

Problem (ARP). Let $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{R})$ be a connected graph without loops, where $\boldsymbol{V} = \{v_1, \ldots, v_n\}$ is the vertex set, representing the street intersections and dead-ends of the estate; $\boldsymbol{E} = \{(v_i, v_j) : v_i, v_j \in \boldsymbol{V} \text{ and } i < j\}$ is the edge set, representing the road segments of the estate; and $\boldsymbol{R} \subseteq \boldsymbol{E}$ representing the road segments that have to be patrolled (traversed) by the guards. Edges that have to be traversed are termed *required* edges, with the remaining edges $\boldsymbol{E}\backslash\boldsymbol{R}$ termed *non-required* edges. The resulting network representation for the estate is given in Figure 2. Every edge (v_i, v_j) is associated with a nonnegative distance or length d_{ij}, assuming that $d_{ij} = \infty$ if (v_i, v_j) is not defined.

The aim of an ARP is defined by Eiselt et al. (1995a) as determining a least-cost traversal of a specified subset of a graph, with or without constraints. Other ARP applications include, for example, the routing of postmen, meter readers and power line inspectors. For a review of ARPs the reader is referred to Corberán and Prins (2010); Wøhlk (2008); Dror (2000); and Eiselt et al. (1995a,b).

Two important ARPs can be derived from general routing problems: the well known Chinese Postman Problem (CPP) and Rural Postman Problem (RPP). For the CPP the complete edge set $\boldsymbol{E}$ has to be traversed by a single postman (or a guard for our application), whereas the RPP requires that only the subset $\boldsymbol{R}$ of edges (with $\boldsymbol{R} \subseteq \boldsymbol{E}$) be traversed. Note that the RPP transforms into the CPP if $\boldsymbol{R} = \boldsymbol{E}$. The objective of both these problems is to find a closed route of minimum length that traverses the required edges. Edmonds and Johnson (1973) show that the CPP can be solved optimally in polynomial time through a matching based algorithm. The RPP is $\mathscr{NP}$-hard (Lenstra and Rinnooy Kan, 1976), but a polynomial solvable case occurs when the graph induced by the required edge set, $\bar{\boldsymbol{G}} = (\boldsymbol{V}, \boldsymbol{R})$, is connected. The problem can then be solved

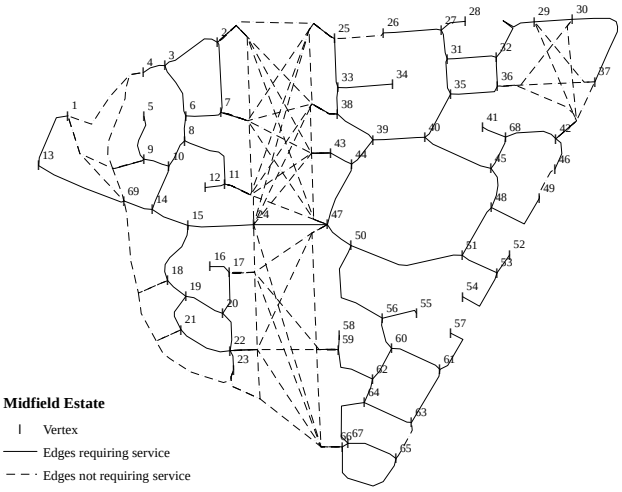

Figure 2: Network representation of Midfield-Estate

by computing shortest chains (in $\boldsymbol{G}$) between odd-degree vertices and then proceeding as for the CPP (Eiselt et al., 1995b).

Both the CPP and RPP can be further expanded to include multiple (k) postmen. The objective is then to either minimise the total distance travelled by the k postmen, subject to a maximum single route length restriction, or to minimise the length of the longest route. The latter are known as the min-max k-RPP and min-max k-CPP. As stated in Ahr and Reinelt (2006) the min-max objective function, abbreviated MM, is ideal when each edge has to be served as early as possible and when more balanced routes are required. Subsequently the problem of designing patrol routes for an estate can best be formulated as either an MM k-CPP, when all edges have to be patrolled, or an MM k-RPP. Both problems have to our knowledge received minimal attention in literature. The most extensive work on the MM k-CPP is by Ahr (2004). Other contributions are by Ahr and Reinelt (2002), Ahr and Reinelt (2006) and Frederickson et al. (1978). The only contributions for the MM k-RPP that we found relevant are by Arkin et al. (2006) and Benevant et al. (2009).

1.2. Solution approaches for min-max multiple Postmen Problems

The MM k-CPP, introduced by Frederickson et al. (1978), and the MM k-RPP are $\mathscr{NP}$-hard. The MM k-CPP was shown $\mathscr{NP}$-hard by a reduction from the k-partition problem (Frederickson et al., 1978), whereas the MM k-RPP with a single guard reduces to the RPP, which is $\mathscr{NP}$-hard. As such, solution strategies for these problems are primarily based on heuristic and metaheuristic methods. Two exceptions are the branch-and-cut algorithm by Ahr (2004) for the MM k-CPP and another branch-and-cut algorithm by Benevant et al. (2009) for the MM k-Windy RPP. The MM k-Windy RPP is an extension of the MM k-RPP where each edge is assigned different traversal costs for the two directions in which the edge can be traversed. Due to the difficulty of the problems the solution approaches of Ahr (2004) and Benevant et al. (2009) failed to solve certain instances of the problems within specified time limits. For patrol guard routing the solution approach must be capable of generating multiple feasible solutions, thus making exact solution approaches impractical.

To our knowledge the only heuristic for the MM k-RPP is the 7-approximation algorithm of Arkin et al. (2006). For the MM k-CPP, Frederickson et al. (1978) developed the Frederickson-Hecht-Kim (FHK) heuristic. Ahr and Reinelt (2002) developed four heuristics for the same problem: an Augment-Merge heuristic, based on the work of Golden and Wong (1981); a Cluster algorithm; as well as two improvement heuristics. Their initial solution heuristics with the improvement procedures outperform the FHK-heuristic on all test instances.

Currently, the only metaheuristic solution algorithm for the MM k-CPP is the tabu search of Ahr and Reinelt (2006). Tabu search is a local search-based solution strategy that starts from an initial feasible solution and progressively attempts to improve it by applying a series of local modifications. At each

iteration the algorithm moves, according to specified criteria, to a feasible solution that differs only slightly from the current one. The search repeats for a fixed number of iterations. What distinguishes it from a greedy local search is that it uses memory to intelligently guide the search. The algorithm deals with cycling by temporarily forbidding moves that would return to recently visited solutions. Although solutions may be revisited after numerous iterations, short-term cycling is prevented. An overview of tabu search is given by Glover (1989, 1990) and a detailed description by Glover and Laguna (1998). To date the best solutions for MM k-CPP test instances are found by the tabu search of Ahr and Reinelt (2006), which uses the solutions of Ahr and Reinelt (2002) as starting point. Tabu search-based solution strategies are also used to solve a similar problem to the MM k-CPP, the Capacitated Arc Routing Problem (CARP).

The CARP, introduced by Golden and Wong (1981), is essentially the k-RPP where each required edge $(v_i, v_j) \in \boldsymbol{R}$ has a nonnegative demand that must be collected by a postman or more accurately, by a vehicle. Required edges in $\boldsymbol{R}$ may still be traversed without service, referred to as *deadheading*, and the sum of demand for serviced edges on any vehicle route may not exceed the vehicle's capacity. Examples of the CARP include the routing of urban waste collection vehicles, street sweepers, and snow removal vehicles. There are numerous successful applications of tabu search to solve the CARP. A tabu search algorithm called CARPET was introduced by Hertz et al. (2000), while Greistorfer (2003) uses a scatter search variant. Other variations on the tabu search include the deterministic tabu search (Brandão and Eglese, 2008); and the use of capacitated trees to solve a Multiple Centre CARP (Amberg et al., 2000). In the next section we build on the successful tabu search contributions, and propose a variation of the algorithm suited for security guard routing.

2. A tabu search algorithm for patrol route generation

In this section we introduce basic terminology and give a high-level description of our proposed algorithm for patrol route generation with reference and detailed descriptions of its embedded procedures.

Our tabu search algorithm is influenced by and contains elements of the algorithm of Ahr and Reinelt (2006) for the MM k-CPP. What distinguishes our work is, firstly, that our algorithm is designed to be applied to both the MM k-CPP and MM k-RPP, whereas the algorithm of Ahr and Reinelt (2006) is designed for the MM k-CPP. In their implementation all edges that form part of a route are explicitly modelled and subjected to removal and insertion procedures. An improvement procedure is further used to check if edges are unnecessarily traversed in a route, i.e., they are already traversed in another route, in which case they are removed. This approach works well if all the edges are required, as with the MM k-CPP, but the MM k-RPP consists of required and nonrequired edges and applying the procedures to nonrequired edges are ineffective. Our algorithm uses the encoding scheme of Lacomme et al. (2004) for the CARP in which only required edges are explicitly modelled per route and it is assumed that the shortest path, consisting of required and non-required edges, is always followed between consecutive required edges. Applying any sort of removal and insertion procedure to a nonrequired edge will only worsen a route, hence the procedures are only applied to required edges. The second distinguishing factor is that our algorithm uses a more involved improvement procedure and different neighbourhood constructors that are more appropriate to the MM k-RPP. Lastly, our algorithm is capable of generating multiple solutions for the same problem instance. Different solutions are essential for our application since guard patrolling has to be unpredictable and requires different routes.

Our algorithm, called Tabu-Guard, works in three phases. During the first phase it generates a specified number of different initial solutions using a constructive heuristic called Generate-Random-Initial-Solutions. In the second phase it improves the initial solutions by calling Improve-Solutions, and during the third phase it improves the solutions further by calling on a tabu search algorithm simply called Tabu-Search.

2.1. Basic terminology and algorithm encoding scheme

For the MM k-RPP and MM k-CPP we have the following input data: a connected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{R})$, length or distance matrix $\boldsymbol{D} = \{d_{ij}\}$, a distinguished depot vertex v_1 and a fixed number of $k > 1$ guards; for our application we refer to guards instead of postmen. Consistent with the work of Lacomme et al. (2004) for the CARP the graph $\boldsymbol{G}$ is transformed into a fully directed graph $\boldsymbol{G'} = (\boldsymbol{V}, \boldsymbol{A'}, \boldsymbol{R'})$ by replacing each edge $(v_i, v_i) \in \boldsymbol{R}$ with two opposite arcs $\{(v_i, v_j), (v_j, v_i)\} \in \boldsymbol{A'}$, both with the same traversal cost. Arcs in $\boldsymbol{A'}$ are indentified by indices from 1 to m where $m = |\boldsymbol{A'}|$ and the traversal cost of arc u is given by $c(u)$. The required edges $\boldsymbol{R}$ correspond in $\boldsymbol{G'}$ to a subset of required arcs $\boldsymbol{R'} \subseteq \boldsymbol{A'}$, such that $|\boldsymbol{R'}| = 2|\boldsymbol{R}|$. Each

required arc u has a pointer $inv(u)$ corresponding to the inverse arc of u, which is the second orientation of the original edge. Thus if u and v represent opposite arcs (v_k, v_l) and (v_l, v_k), respectively, then $inv(u) = v$, $inv(v) = u$ and $c(u) = c(v)$. Lastly, the depot is modelled by including in $\boldsymbol{A}'$ a fictitious loop $\sigma = (v_1, v_1)$, with $c(\sigma) = 0$ and $inv(\sigma) = \sigma$. Since $\boldsymbol{G}'$ is directed, we refer to required and nonrequired *arcs* in the remainder of this section, where each arc represents one of the traversal directions of the corresponding edge.

A feasible solution for the MM k-RPP or MM k-CPP is a set $\boldsymbol{T}$ of k closed routes $\boldsymbol{T} = \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$ such that each route $\boldsymbol{C}_i$ contains the depot arc σ and all the required edges $\boldsymbol{R}$ (where $\boldsymbol{R} = \boldsymbol{E}$ for the CPP variant) are covered by at least one route $\boldsymbol{C}_i$. With the chosen encoding scheme either arc u or $inv(u) \in \boldsymbol{R}'$ must be covered by a route. Accordingly, if arc u is assigned to a route, $inv(u)$ is automatically marked as covered. A route $\boldsymbol{C}_i$ is a string of arc indices, which, in turn, represent arcs that the guard visits in sequence. With this representation $\boldsymbol{C}_i(t)$ is defined as the arc in position t in route $\boldsymbol{C}_i$. We define a distance function w, where $w(\boldsymbol{C}_i)$ is the total length of route $\boldsymbol{C}_i$. For a feasible solution, denote the distance of the longest single route $\boldsymbol{C}_i$ as $w_{max}(\boldsymbol{T})$, determined by:

$$w_{max}(\boldsymbol{T}) = \max_{i=1,\ldots,k} w(\boldsymbol{C}_i).$$

The objective of the MM k-RPP and MM k-CPP is to find a solution $\boldsymbol{T}^*$ that minimises w_{max} among all feasible solutions.

Finally, denote by $\boldsymbol{SP}(u, v)$ the set of arcs on the shortest path between but excluding arcs u and v. The distance of such a path is given by $\boldsymbol{D}_{SP}(u, v)$, which again excludes the traversal cost of u and v. The Shortest-Path between all arcs can be efficiently pre-computed using an adaption of Dijkstra's shortest path algorithm (Lacomme et al., 2004). We further denote by $\boldsymbol{SP}_R(u, v)$ the set of *required* arcs that form part of the shortest path between u and v. Lastly we denote by $\boldsymbol{Q}(\boldsymbol{C}_i, \boldsymbol{T})$ the set of required arcs that form part of the shortest paths within route $\boldsymbol{C}_i$ and that are currently not assigned to any route in solution $\boldsymbol{T}$. This can be easily determined using the current solution $\boldsymbol{T}$, route $\boldsymbol{C}_i$ and $\boldsymbol{SP}_R$.

2.2. Phase 1: Generating random initial solutions

The first phase of Tabu-Guard entails generating a multitude of different initial solutions using Generate-Random-Initial-Solutions. To create a single feasible solution, the algorithm first creates $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$ routes by finding the untraversed required arcs that are furthest from the depot (guard house), and creating k closed routes that traverse these arc.

Initially let $\boldsymbol{R}'' = \boldsymbol{R}'$. The arc that is the furthest from the depot is determined through

$$u = \arg\max\{\boldsymbol{D}_{SP}(\sigma, v) + c(v) + \boldsymbol{D}_{SP}(v, \sigma) : v \in \boldsymbol{R}''\}.$$

The required arcs that are traversed in the shortest paths from σ to u and from u back to σ, given by $\boldsymbol{SP}_R(\sigma, u)$ and $\boldsymbol{SP}_R(u, \sigma)$, respectively, are spliced together to form a closed route

$$\boldsymbol{C}_i = \{\sigma, \boldsymbol{SP}_R(\sigma, u), u, \boldsymbol{SP}_R(u, \sigma), \sigma\}.$$

All the required arcs in route $\boldsymbol{C}_i$, together with their inverse arcs, are then removed from $\boldsymbol{R}''$. The route is scanned and if there are two entries for a required arc in the same route, the second entry is removed. During the scan a required arc is also removed if it is already assigned to another route, thus if the arc is not in $\boldsymbol{R}''$. This process is repeated k times, resulting in k closed routes.

The resulting solution is most likely still infeasible, so next the algorithm iteratively adds required arcs not yet serviced to the existing routes. The algorithm produces a random order of $\boldsymbol{R}''$ and the first entry is temporarily added to each of the $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$ routes through the procedure Insert-Arc (Appendix A, Algorithm 6). The procedure adds an arc u to $\boldsymbol{C}_i$ in position t, which is always between the begin and end depot arcs, that results in the minimum route cost increase. The procedure also determines the best orientation by testing the insertion of both u and $inv(u)$ in each position. The temporary route that is the least affected by the insertion, i.e., the route with the least cost increase, is then made permanent.

Once a required arc u has been added to a route $\boldsymbol{C}_i$, all newly traversed required arcs in $\boldsymbol{C}_i$ are determined through $\boldsymbol{Q}(\boldsymbol{C}_i, \boldsymbol{T})$ and added in their current positions to the route. These required arcs and arc u, together with their inverse arcs are then removed from $\boldsymbol{R}''$. The route is then scanned for duplicate arc entries and the second entries are removed. The process is then repeated with the new first arc from $\boldsymbol{R}''$ and the process terminates when $\boldsymbol{R}''$ is empty and all required arcs are traversed, with $\boldsymbol{T}$ the resulting solution.

Importantly, each required arc or its inverse is assigned to only one route. Since a different sequence for the required edges in $\boldsymbol{R}''$ will be generated each time this process is invoked, GENERATE-INITIAL-SOLUTIONS is capable of generating multiple initial solutions. The complete pseudo code for the algorithm is presented in Algorithm 1.

Algorithm 1: GENERATE-RANDOM-INITIAL-SOLUTIONS

Input : Number of guards k and number of solutions to generate n.
Output: Multiple feasible initial solutions $\{\boldsymbol{T}^1, \ldots, \boldsymbol{T}^n\}$.

for $i \leftarrow 1$ **to** n **do**
  $\boldsymbol{R}'' \leftarrow \boldsymbol{R}'$;
  for $j \leftarrow 1$ **to** k **do**
    Let $u \leftarrow \arg\max\{\boldsymbol{D}_{SP}(\sigma, v) + c(v) + \boldsymbol{D}_{SP}(v, \sigma) : v \in \boldsymbol{R}''\}$;
    $\boldsymbol{C}^i_j \leftarrow \{\sigma, \boldsymbol{SP}_R(\sigma, u), u, \boldsymbol{SP}_R(u, \sigma), \sigma\}$;
    Remove from $\boldsymbol{C}^i_j$ all arcs not in $\boldsymbol{R}''$, and if an arc is in the route more than once, either in its original or inverse orientation, remove the second entry;
    Remove from $\boldsymbol{R}''$ all the newly traversed arcs, including u, and their inverse arcs;
  Generate a random order of the remaining untraversed required arcs $\boldsymbol{R}''$;
  while $\boldsymbol{R}'' \neq \emptyset$ **do**
    Let u be the first entry in $\boldsymbol{R}''$;
    for $j \leftarrow 1$ **to** k **do**
      $\boldsymbol{C}'_j \leftarrow$ INSERT-ARC$(\boldsymbol{C}^i_j, u)$;
      $\Delta\Theta_j \leftarrow w(\boldsymbol{C}'_j) - w(\boldsymbol{C}^i_j)$
    Find $\boldsymbol{C}'_j$ where $j \leftarrow \arg\min\{\Delta\Theta_t : t \in (1, \ldots, k)\}$;
    Let $\boldsymbol{C}^i_j \leftarrow \boldsymbol{C}'_j$ and $\boldsymbol{T}' \leftarrow \{\boldsymbol{C}^i_1, \ldots, \boldsymbol{C}^i_k\}$;
    Let $\boldsymbol{L} \leftarrow \boldsymbol{Q}(\boldsymbol{C}^i_j, \boldsymbol{T}')$ and add all the arcs in $\boldsymbol{L}$ to route $\boldsymbol{C}^i_j$;
    Remove u, the arcs in $\boldsymbol{L}$ and their respective inverse arcs from $\boldsymbol{R}''$;
    Scan $\boldsymbol{C}^i_j$ and if an arc is in the route more than once, either in its original or inverse orientation, remove the second entry;
  $\boldsymbol{T}^i = \{\boldsymbol{C}^i_1, \ldots, \boldsymbol{C}^i_k\}$;
return $(\{\boldsymbol{T}^1, \ldots, \boldsymbol{T}^n\})$

2.3. Phase 2: Improving the initial solutions

Given the simplistic nature of GENERATE-RANDOM-INITIAL-SOLUTIONS the initial solutions are usually excessively long. In response we developed IMPROVE-SOLUTION and IMPROVE-SINGLE-ROUTE. IMPROVE-SOLUTION incrementally tries to improve an initial solution $\boldsymbol{T}$ by improving each of its guard routes $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$ in such a way that longest route is improved the most.

IMPROVE-SOLUTION works in two phases. First it lets $\boldsymbol{R}'' = \boldsymbol{R}'$ and sorts $\boldsymbol{T}$ from the shortest to longest route. It then takes the shortest route $\boldsymbol{C}_1$ and uses $\boldsymbol{SP}_R$ to determine *all* the required arcs that are traversed in $\boldsymbol{C}_1$, regardless if they are assigned to other routes. The required arcs are then formally assigned to $\boldsymbol{C}_1$ and removed from $\boldsymbol{R}''$, together with their inverse arcs. The same is then applied to the second shortest route $\boldsymbol{C}_2$, but required arcs are only assigned to $\boldsymbol{C}_2$ if they are in $\boldsymbol{R}''$, after which they are removed from $\boldsymbol{R}''$. The process is repeated for the remaining routes and finishes with $\boldsymbol{C}_k$. The net result is that the minimum number of required arcs are assigned to the longest route $\boldsymbol{C}_k$. Since shortest paths are always followed between assigned arcs, the cost of the routes are usually reduced, and at worst stays the same.

In the second phase IMPROVE-SOLUTION tries to find a better sequence in which each route's assigned arcs are traversed. Two move procedures are used for this purpose. The first, EXCHANGE-ARCS (Appendix A, Algorithm 7), takes $\boldsymbol{C}_i$ and performs a pair-wise exchange in the sequence in which the assigned arcs are traversed. The pair-wise exchange is performed between all arcs in $\boldsymbol{C}_i$ and the algorithm also determines the best orientation for the exchanged arcs.

The second procedure, REMOVE-INSERT-ARCS (Appendix A, Algorithm 8), simply removes arc v from $\boldsymbol{C}_i$ and uses INSERT-ARC to determine if the arc can be inserted in a better position in the route. REMOVE-INSERT-ARCS is also applied to each arc in $\boldsymbol{C}_i$. Both procedures return the route resulting from the best

move. The route $\boldsymbol{C}_i$ is then set to the best of the two routes returned, but only if it is better than original $\boldsymbol{C}_i$. Exchange-Arcs and Remove-Insert-Arcs are then again applied to $\boldsymbol{C}_i$. If no improving move can be made, the algorithm tries to improve the next route in $\boldsymbol{T}$. The second phase terminates when none of the routes can be improved any further.

If any route was improved in the second phase, Improve-Solution returns to the first phase and the improvement process is repeated, otherwise the algorithm terminates. The pseudo code for Improve-Solution is shown in Algorithm 2. Improve-Solution is also imbedded in Tabu-Search, which we describe next.

Algorithm 2: Improve-Solution

Input : Initial solution $\boldsymbol{T} = \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$.
Output: Improved solution $\boldsymbol{T}$.

repeat
  $Improve \leftarrow False$;
  Sort $\boldsymbol{T}$ from the shortest to the longest route. Let $\boldsymbol{R}'' \leftarrow \boldsymbol{R}'$ and $\boldsymbol{T}^{temp} \leftarrow \emptyset$;
  for $i \leftarrow 1$ **to** k **do**
    foreach $u \in \boldsymbol{C}_i$ **do if** $u \notin \boldsymbol{R}''$ **then** remove u from $\boldsymbol{C}_i$;
    Add $\boldsymbol{C}_i$ to $\boldsymbol{T}^{temp}$;
    $\boldsymbol{L} \leftarrow \boldsymbol{Q}(\boldsymbol{C}_i, \boldsymbol{T}^{temp})$;
    Add all the arcs in $\boldsymbol{L}$ to $\boldsymbol{C}_i$;
    Remove the arcs in $\boldsymbol{C}_i$ and their inverse arcs from $\boldsymbol{R}''$;
  for $i \leftarrow 1$ **to** k **do**
    repeat
      $RouteImproved \leftarrow False$;
      $\boldsymbol{C}'_1 \leftarrow$ Exchange-Arcs($\boldsymbol{C}_i$);
      $\boldsymbol{C}'_2 \leftarrow$ Remove-Insert-Arcs($\boldsymbol{C}_i$);
      if $\min\{w(\boldsymbol{C}'_1), w(\boldsymbol{C}'_2)\} < w(\boldsymbol{C}_i)$ **then**
        Let $RouteImproved \leftarrow True$ and $Improve \leftarrow True$;
        Let $\boldsymbol{C}_i \leftarrow \boldsymbol{C}'_j$ where $j \leftarrow \arg\min\{w(\boldsymbol{C}'_j) : j \in (1, 2)\}$;
    until $RouteImproved = False$;
  $\boldsymbol{T} \leftarrow \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$;
until $Improve = False$;
return $(\boldsymbol{T})$

2.4. Phase 3: Tabu search

During the last phase of Tabu-Guard the algorithm calls Tabu-Search to further improve the initial solutions. Tabu-Search starts by generating multiple neighbourhood solutions by performing modification procedures on an initial solution. The algorithm then moves to the best non-tabu solution (the solution is in effect chosen for the next iteration). The chosen solution does not have to be better than the previous one, it merely has to be the best neighbour. During the search the algorithm continuously keeps track of and updates the best solution found, referred to as the incumbent solution . Since worse solutions are chosen the algorithm avoids getting stuck at a local optimum. However, without governing the search the algorithm may cycle by moving back to already visited solutions, including the local optimum. To prevent short-term cycling, the algorithm uses a tabu list that temporarily stores information pertaining to neighbourhood moves recently made. If characteristics of a potential neighbourhood move are consistent with information stored on the list, the move is considered tabu and cannot be chosen. There is, however, one exception to this rule. Since the list only stores information of the recent *moves* made to construct solutions, and does not contain the complete solutions, a new solution not previously visited may still be constructed using a tabu move. Consequently, a tabu move may be chosen and implemented, but only if it results in a new incumbent solution.

2.4.1. Constructing neighbourhood solutions

Tabu-Search constructs neighbourhood solutions by removing edges from one route, and adding them to another. Two modification procedures, Remove-Insert-Neighbourhood and Exchange-Neighbourhood,

are used for this addition and removal. As their names suggest, they are extensions of Remove-Insert-Arcs and Exchange-Arcs. The modification procedures are applied to two routes at a time: the longest route $\boldsymbol{C}_i$ and any other route $\boldsymbol{C}_j$. Remove-Insert-Neighbourhood generates neighbouring solutions by taking each arc in route $\boldsymbol{C}_i$, removing it from the route and inserting it in route $\boldsymbol{C}_j$ through Insert-Arc (Appendix A, Algorithm 6). Similarly, Exchange-Neighbourhood removes an arc u from $\boldsymbol{C}_i$, but it then also removes an arc v from $\boldsymbol{C}_j$. It then calls Insert-Arc and inserts v in $\boldsymbol{C}_i$ and u in $\boldsymbol{C}_j$. The process is repeated for each two-arc combination with u in $\boldsymbol{C}_i$ and v in $\boldsymbol{C}_j$. The overall best neighbouring solution and the best non-tabu neighbouring solution are then improved with Improve-Solution and returned. The procedures are presented in Algorithms 3 and 4. Note that the test to check if a move is tabu, Tabu-Test, is described in the latter part of the section. Tabu-Search uses one or both of the modification procedures and moves to the best solution returned that is non-tabu, or to the overall best solution if it is better than the current incumbent solution.

Algorithm 3: Remove-Insert-Neighbourhood

Input : Solution $\boldsymbol{T} = \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$.
Output: Overall best neighbouring solution $\hat{\boldsymbol{T}}_{all}$ and best non-tabu neighbouring solution $\hat{\boldsymbol{T}}_{nt}$.

Sort $\boldsymbol{T}$ from the longest to the shortest route and let $\hat{\boldsymbol{T}}_{nt} \leftarrow \boldsymbol{T}$;
Let $w_1^* \leftarrow \infty$ and $w_2^* \leftarrow \infty$;
foreach $u \in \boldsymbol{C}_1$ **do**
  Let $\boldsymbol{C}'_1 \leftarrow \boldsymbol{C}_1$ and remove u from $\boldsymbol{C}'_1$;
  for $i \leftarrow 2$ **to** k **do**
    $\boldsymbol{C}'_i \leftarrow$ Insert-Arc$(\boldsymbol{C}_i, u)$;
    $Tabu \leftarrow$ Tabu-Test$(u, \boldsymbol{C}_1, \boldsymbol{C}_i)$;
    if $\max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\} < w_1^*$ **then**
      $w_1^* \leftarrow \max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\}$;
      $\hat{\boldsymbol{T}}_{all} \leftarrow \{\boldsymbol{C}'_1, \ldots, \boldsymbol{C}'_k\}$;
    if $\max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\} < w_2^*$ *and* $Tabu = False$ **then**
      $w_2^* \leftarrow \max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\}$;
      $\hat{\boldsymbol{T}}_{nt} \leftarrow \{\boldsymbol{C}'_1, \ldots, \boldsymbol{C}'_k\}$;
  $\boldsymbol{C}'_i \leftarrow \boldsymbol{C}_i$;
$\hat{\boldsymbol{T}}_{all} \leftarrow$ Improve-Solution$(\hat{\boldsymbol{T}}_{all})$;
$\hat{\boldsymbol{T}}_{nt} \leftarrow$ Improve-Solution$(\hat{\boldsymbol{T}}_{nt})$;
return $(\hat{\boldsymbol{T}}_{all}, \hat{\boldsymbol{T}}_{nt})$

2.4.2. Tabu list and stopping criteria

After moving to a neighbourhood solution, Tabu-Search adds solution criteria of the chosen neighbourhood solution to the tabu list. The criteria are then removed from the tabu list after a certain number of iterations, referred to as the tabu tenure, α, has passed. We developed three different tabu list strategies that use different criteria to determine if a move is a tabu. The first, Complex-Tabu, which is also used by Ahr and Reinelt (2006), uses the following criteria: the arc u removed route from $\boldsymbol{C}_i$; the route i from which it was removed; and the route j to which the arc was added. Since a solution's routes are continuously sorted during the search, the original position of the route is saved prior to the execution of Tabu-Search. Subsequently, the original position of $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, given by the functions $o(\boldsymbol{C}_i)$ and $o(\boldsymbol{C}_j)$, instead of the current positions, i and j, are used by the strategy. In the following iterations a neighbouring solution constructed by removing arc u or $inv(u)$ from route $o(\boldsymbol{C}_i)$ or route $o(\boldsymbol{C}_j)$ and adding them to $o(\boldsymbol{C}_j)$, if removed from $o(\boldsymbol{C}_i)$, or adding them to $o(\boldsymbol{C}_i)$, if removed from $o(\boldsymbol{C}_j)$, will constitute a tabu neighbourhood solution.

The second strategy, Simple-Tabu, is simpler and more aggressive. The only criteria used is the arc u removed from route $\boldsymbol{C}_i$. Any other move that involves arc u or $inv(u)$ constitutes a tabu neighbouring solution. The third strategy, Simple-Aggressive-Tabu, is the same as Simple-Tabu, but it also enforces the tabu criteria in the first phase of Improve-Solution by prohibiting tabu arcs from being removed from their current routes.

Tabu-Search terminates when all the neighbourhood solutions are tabu, and the best tabu solution does not improve on the incumbent solution. The algorithm may also terminate if the incumbent solution has

Algorithm 4: EXCHANGE-NEIGHBOURHOOD

Input : Solution $\boldsymbol{T} = \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k\}$.
Output: Overall best neighbouring solution $\hat{\boldsymbol{T}}_{all}$ and best non-tabu neighbouring solution $\hat{\boldsymbol{T}}_{nt}$.

Sort $\boldsymbol{T}$ from the longest to the shortest route and let $\hat{\boldsymbol{T}}_{nt} \leftarrow \boldsymbol{T}$;
Let $w_1^* \leftarrow \infty$ and $w_2^* \leftarrow \infty$;
foreach $u \in \boldsymbol{C}_1$ **do**
  Let $\boldsymbol{C}'_1 \leftarrow \boldsymbol{C}_1$ and remove u from $\boldsymbol{C}'_1$;
  for $i \leftarrow 2$ **to** k **do**
    foreach $v \in \boldsymbol{C}_i$ **do**
      Let $\boldsymbol{C}'_i \leftarrow \boldsymbol{C}_1$ and remove v from $\boldsymbol{C}'_i$;
      Let $\boldsymbol{C}'_1 \leftarrow$ INSERT-ARC$(\boldsymbol{C}'_1, v)$ and $\boldsymbol{C}'_i \leftarrow$ INSERT-ARC$(\boldsymbol{C}'_i, u)$;
      Let $Tabu1 \leftarrow$ TABU-TEST$(u, \boldsymbol{C}_1, \boldsymbol{C}_i)$ and $Tabu2 \leftarrow$ TABU-TEST$(v, \boldsymbol{C}_i, \boldsymbol{C}_1,)$;
      if $\max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\} < w_1^*$ **then**
        $w_1^* \leftarrow \max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\}$;
        $\hat{\boldsymbol{T}}_{all} \leftarrow \{\boldsymbol{C}'_1, \ldots, \boldsymbol{C}'_k\}$;
      if $\max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\} < w_2^*$ *and* $Tabu1 = False$ *and* $Tabu2 = False$ **then**
        $w_2^* \leftarrow \max\{\boldsymbol{C}'_1, \boldsymbol{C}'_i\}$;
        $\hat{\boldsymbol{T}}_{nt} \leftarrow \{\boldsymbol{C}'_1, \ldots, \boldsymbol{C}'_k\}$;
$\hat{\boldsymbol{T}}_{all} \leftarrow$ IMPROVE-SOLUTION$(\hat{\boldsymbol{T}}_{all})$;
$\hat{\boldsymbol{T}}_{nt} \leftarrow$ IMPROVE-SOLUTION$(\hat{\boldsymbol{T}}_{nt})$;
return $(\hat{\boldsymbol{T}}_{all}, \hat{\boldsymbol{T}}_{nt})$

not been improved for a certain number of iterations, t_{max}. The complete pseudo code for TABU-SEARCH is presented in Algorithm 5.

To generate patrol routes TABU-GUARD executes the following three steps. First it generates multiple random initial solutions by calling GENERATE-RANDOM-INITIAL-SOLUTIONS. Next it uses IMPROVE-SOLUTION to improve each initial solution and then, lastly, it uses TABU-SEARCH to improve the solutions even further.

3. Lower bounds

To assess the solution quality of the routes generated with TABU-GUARD we use three lower bounds from Ahr and Reinelt (2002).

The first lower bound considered is the *Shortest Path Tour Lower Bound* (SPT-LB). In the optimal solution the required arc, u, that is the furthest away from the depot arc, σ, must be traversed by one of the k guard routes. The longest route must have at least the length of the shortest route, $(\boldsymbol{SP}(\sigma, u), u, \boldsymbol{SP}(u, \sigma))$, traversing the arc furthest from the depot.

The second lower bound is the *CPP Tour Lower Bound* (CPP/k-LB) and is computed by finding the optimal Chinese postman route, and dividing its weight by k. We only consider the bound for the MM k-CPP, and for the MM k-RPP if the required edges $\boldsymbol{R}$ form a connected subgraph. MM k-RPP instances with $\boldsymbol{R}$ disconnected would involve finding the optimal rural postman tour, thus solving the RPP which is $\mathcal{NP}$-hard.

The last lower bound is the *IP Relaxation Lower Bound* (IP-LB). The bound is computed by solving a relaxed integer programming formulation for the MM k-RPP. For simplicity we define $e = (v_i, v_j)$ where $(v_i, v_j) \in \boldsymbol{E}$, v_i and $v_j \in \boldsymbol{V}$, and $i < j$. We define L_{max} as the total distance of the longest guard route. The decision variables for the MM k-RPP are

$$x_i(e) \triangleq \begin{cases} 1 & \text{if edge } e \text{ is serviced by route } \boldsymbol{C}_i, \text{ where } i = \{1, \ldots, k\} \text{ and } e \in \boldsymbol{R}, \\ 0 & \text{otherwise,} \end{cases}$$

$y_i(e) \triangleq$ Number of times edge e is traversed by route $\boldsymbol{C}_i$ without being serviced, where $i = \{1, \ldots, k\}$ and $e \in \boldsymbol{E}$.

Other model parameters are

Algorithm 5: TABU-SEARCH

Input : Starting solution $\tilde{\boldsymbol{T}}$; neighbourhood construction procedure to use CONSTRUCT (can either be EXCHANGE-NEIGHBOURHOOD, REMOVE-INSERT-NEIGHBOURHOOD or both); max number of iterations without improvement t_{max}; tabu tenure α; and a tabu list strategy TABU-STRATEGY (can either be COMPLEX-TABU, SIMPLE-TABU or SIMPLE-AGGRESSIVE-TABU).

Output: Incumbent solution $\boldsymbol{T}^*$.

Let $\boldsymbol{T}^* \leftarrow \boldsymbol{T}$ and $w^*_{max} \leftarrow w_{max}(\boldsymbol{T})$;
Let the tabu list be $\boldsymbol{\Pi}$, which is initially empty;
$t \leftarrow 0$;
while $t < t_{max}$ **do**
  Remove move criteria from $\boldsymbol{\Pi}$ that have been on the list for more than α iterations;
  $(\hat{\boldsymbol{T}}_{all}, \hat{\boldsymbol{T}}_{nt}) \leftarrow$ CONSTRUCT($\boldsymbol{T}$);
  if $w_{max}(\hat{\boldsymbol{T}}_{all}) < w^*_{max}$ **then**
    $t \leftarrow 0$;
    $w^*_{max} \leftarrow w_{max}(\boldsymbol{T}_{all})$;
    $\boldsymbol{T}^* \leftarrow \boldsymbol{T}_{all}$;
  else if $\hat{\boldsymbol{T}}_{nt} \neq \boldsymbol{T}$ **then**
    $t \leftarrow t + 1$;
    $\boldsymbol{T} \leftarrow \hat{\boldsymbol{T}}_{nt}$;
    According to TABU-STRATEGY add the appropriate move criteria to $\boldsymbol{\Pi}$;
  else $t \leftarrow t_{max}$
return $(\boldsymbol{T}^*)$

$d(e) \triangleq$ Length or cost of edge e, where $e \in \boldsymbol{E}$.

For the MM k-RPP we have the following mathematical model

$$\min z = L_{max} \tag{1}$$

subject to

$$\sum_{i=1}^{k} x_i(e) = 1 \qquad \forall\, e \in \boldsymbol{R}, \tag{2}$$

$$\sum_{e \in \boldsymbol{E}} d(e) y_i(e) + \sum_{e \in \boldsymbol{R}} d(e) x_i(e) \leq L_{max} \qquad \forall\, i = \{1, \ldots, k\}, \tag{3}$$

$$\sum_{e \in \delta(v)} x_i(e) + y_i(e) \equiv 0 \pmod 2 \qquad \forall\, v \in \boldsymbol{V}, i = \{1, \ldots, k\}, \tag{4}$$

$$x_i(\delta(\boldsymbol{S})) + y_i(\delta(\boldsymbol{S})) \geq 2x_i(e) \qquad \forall\, \boldsymbol{S} \subseteq \boldsymbol{V} \backslash \{v_1\}, e \in \boldsymbol{E}(\boldsymbol{S}), i = \{1, \ldots, k\}, \tag{5}$$

$$x_i \in \{0, 1\}, y_i(e) \geq 0 \text{ and integer} \qquad \forall e \in \boldsymbol{E}, i = \{1, \ldots, k\}. \tag{6}$$

Constraints (2) ensure that all the required edges are serviced, thus traversed exactly once by a guard. The length of the longest guard route, L_{max}, is captured through constraints (3). Note that its value is calculated based on the decision variables $x_i(e)$ and $y_i(e)$. Each guard route must be a closed walk containing the depot, which is enforced with constraints (4) and (5). For the lower bound, we solve the relaxation of the proposed model by omitting (5).

In the next section we illustrate how TABU-GUARD was used to generate patrol routes for an actual security estate.

4. An illustrative case: developing routes for Midfield Estate using Tabu-Guard

Midfield-Estate (Figure 1) forms part of the greater Midrand-Estates situated in Gauteng, South Africa. In terms of size Midfield-Estate is fairly large and contains 404 properties, a golf course, and a cricket

ground. The road network (Figure 2) of the estate consists of 68 vertices and 126 edges, of which 74 are required edges and 52 non-required edges. The 24-hour patrolling of the estate was divided into ten three-guard shifts with the same three patrol routes traversed per shift. During the analysis of the patrol routes certain major deficiencies were identified. Figure 3 indicates the frequency of traversals of the various road segments with the old patrol routes. Some of the street segments were not being patrolled by the routes,

Figure 3: Number of street traversals through 24-hour patrolling for three guards with the old patrol routes

and are indicated by the black segments. In contrast, other segments were being over-patrolled as much as fifty times during a day, indicated in the figure by the white segments. The structure of the estate's road network and the decentralised location of the depot partly contributed to the over patrolling of certain edges. However, the main contributor was that the guards were forced to only traverse required edges, thus resulting in the edges close to the depot being over patrolled, whereas more balanced patrol routes can be generated by allowing required and non-required edges to form part of a guard's patrol tour. Another deficiency was the large difference between the length of the longest and shortest of the three routes. The longest route was 4624m and the shortest only 3051m with a difference of 1573m. Lastly, the patrolling of the estate was too predictable and inflexible since the same three patrol routes were being followed per shift and the routes assumed that there would always be exactly three guards available. Management indicated that guard availability varied per shift from two to six guards.

4.1. Lower bounds

Before illustrating how new routes were generated we first calculate the three MM k-RPP lower bounds for the range $k = 2, \ldots, 6$ available guards. The lower bounds are used for solution evaluation and are reported in Table 1. All results are given in meters and the tightest bounds shown in bold. Even though the network consists of non-required edges, CPP/k-LB is valid since the required edges form a connected subgraph. For $k \leq 4$, IP-LB performs best (tightest), while SPT-LB dominates the other bounds for $k = 5, 6$.

In the remainder of this section Lower Bound (LB) always refers to the tightest of the three lower bounds for each k. All algorithms were coded in Python version 2.6 and run on a 3 GHz Intel(R) Core(TM)2 Duo CPU with 3.25 GB of RAM, and all computations were performed on the range $k = 2, \ldots, 6$ available guards.

4.2. Computational results and analysis

For the case study the Tabu-Guard algorithm was used to generate a pool of high quality patrol routes, enabling the security manager to randomly choose which patrol routes to implement during a shift while

Table 1: Lower bounds for Midfield-Estate road network.

k	SPT-LB (m)	CPP/k-LB (m)	IP-LB (m)
2	2295	4288	**4401**
3	2295	2858	**3009**
4	2295	2144	**2365**
5	**2295**	1715	2002
6	**2295**	1429	1620

taking in consideration guard availability. The routes were generated as follows. First Tabu-Guard generated nine different initial solutions for each k by calling Generate-Random-Initial-Solutions and improving them with Improve-Solution. The solutions were evaluated based on the longest route distance w_{max} and the lower bound gap, which is calculated as $[w_{max}(\tilde{\boldsymbol{T}}_0) - LB]/[w_{max}(\tilde{\boldsymbol{T}}_0)]$. As shown in Table 2, Generate-Random-Initial-Solutions was able to generate and improve 45 different solutions in a total time of 0.33 seconds. For each k the total time of generating and improving nine initial solutions was always less than

Table 2: Summary results for nine initial solutions for $k = 2, \ldots, 6$ generated and improved by Tabu-Guard for Midfield-Estate. The nine initial solutions for k guards were generated with Generate-Random-Initial-Solutions and each was then improved with Improve-Solution. The reported CPU time is the total time of generating and improving the nine solutions.

	Best solution		Worst solution		Average	Total
k	w_{max} (m)	LB gap (%)	w_{max} (m)	LB gap (%)	LB gap (%)	CPU time (s)
2	5635	21.90	6584	33.16	29.27	0.11
3	3728	19.38	4336	30.60	24.31	0.06
4	3112	24.00	3290	28.12	25.71	0.06
5	3081	25.51	3267	29.75	27.24	0.05
6	3070	25.24	3377	32.04	27.56	0.05
Total						*0.33s*

0.1 seconds, giving an average time per solution of less than 0.01 seconds. However, the large LB gaps observed, in spite of the application of Improve-Solution, and the large differences between the best and worst solutions indicated that further improvement was possible.

Next Tabu-Guard called Tabu-Search to improve the initial solutions. Tabu-Search has two parameters: the tabu tenure, α, and the number of iterations without improvement, t_{max}. For all our experiments the latter was fixed at 500 iterations. There is also a choice of using one of three neighbourhood exchange procedures: Remove-Insert-Neighbourhood (RIN); Exchange-Neighbourhood (EN); and RIN and EN combined, simply referred to as RINEN. There is then a further choice between using Complex-Tabu, Simple-Tabu and Simple-Aggressive-Tabu as tabu list criteria. To determine the best tabu search setup and tabu criteria, the different setup combinations were tested over the nine initial solutions for each k, with the tabu tenure ranged from $\alpha = \{1, 2, \ldots, 16\}$.

With the experiments the best performances of EN, RIN and RINEN always fell within a small range of α values, irrespective of the tabu criteria used. The best range for RIN and RINEN was $\alpha = \{4, 5, 6\}$ and the best for EN was $\alpha = \{6, 7, 8\}$. Results further showed that EN performs best with Simple-Tabu, whereas both RIN and RINEN perform best with Simple-Aggressive-Tabu. Subsequent analysis and the results reported in Table 3 are limited to these combinations.

In the table, LB gaps are calculated as $[w_{max}(\tilde{\boldsymbol{T}}^i) - LB]/[w_{max}(\tilde{\boldsymbol{T}}^i)]$ and the reported minimum and average values are taken over all 27 solutions (nine initial solutions times three different tabu tenures) for each $k = 2, \ldots, 6$. The average LB gaps of the 27 executions were the lowest with RINEN, except for $k = 2$ where EN produced the lowest average LB gaps. RIN performed the worst for all k values, in terms of both the average and best LB gaps; the latter is calculated with the best solution resulting from the 27 Tabu-Search setup executions for each k. EN and RINEN both produced the same best LB gaps for $k = 3, \ldots, 6$, with EN finding a slightly better solution for $k = 2$. Overall, the LB gaps of the solutions found with all three Tabu-Search setups differed by less than 1%, with the exception of $k = 2$ where the difference between RIN and EN was less than 2%. Large lower bound gaps were still observed for $k = 2, \ldots, 5$, especially for $k = 4$, but as pointed out by Ahr and Reinelt (2006), the lower bounds tend to be weak for this range. In

Table 3: Summary results for nine initial Midfield-Estate solutions for $k = 2, \ldots, 6$ improved by Tabu-Search using one of three neighbourhood exchange procedures. The average and minimum values are taken over 27 executions of the Tabu-Search setup.

	Average LB gap (%)			Best LB gap (%)			Average time per solution (s)		
k	RIN	EN	RINEN	RIN	EN	RINEN	RIN	EN	RINEN
2	12.71	9.56	10.86	7.66	6.52	6.94	11.75	114.04	98.83
3	12.32	12.26	12.13	10.84	10.84	10.84	14.14	61.9	84.47
4	19.52	19.07	19.05	18.17	18.14	18.14	15.94	39.35	50.91
5	12.74	12.71	12.62	11.56	11.66	11.66	18.27	31.47	54.91
6	6.80	6.60	6.66	5.52	5.52	5.52	20.38	21.04	34.34
Average	*12.82%*	*12.04%*	*12.26%*	*10.75%*	*10.54%*	*10.62%*	*16.10s*	*53.56s*	*64.69s*

terms of computational time RIN was the quickest and took, on average, 16.1 seconds to improve a single solution. EN and RINEN were much slower, taking on average 53.6 and 64.7 seconds, respectively, per solution. The differences between execution times were expected since EN and RINEN construct larger solution neighbourhoods than RIN, which also explains why EN and RINEN found better solutions.

4.3. Choosing routes for implementation

With a sufficiently large, good quality solution pool the final step was to choose the actual routes to implement. For each k we evaluated all 81 final solutions—27 solutions from each of the three Tabu-Search setups. The even patrolling of the 81 solutions were measured by taking the difference between the number of times that the most and least traversed edges are traversed, and the five solutions with the smallest differences were chosen for implementation. In case of a tie, the length of the longest route of each solution was calculated, and the first solution with the minimum longest route chosen. Subsequent solutions with the same minimum longest route were then disqualified for selection, ensuring that five unique solutions were always chosen for implementation.

Depending on the guard availability, ranging from two to six, the security manager can now randomly choose which of the five patrol route combinations to implement, the sequence in which they are implemented, and which of the patrol routes should be followed in a clockwise or counter-clockwise direction. Patrolling through the new routes is thus much more unpredictable and flexible than the single three-route combination used previously. Furthermore, Tabu-Guard's solutions outperform the old routes in terms of the length of the longest route and the even work distribution among the guards (Table 4). The latter is measured as the difference between the length of the longest and shortest route. The average length of the old routes is slightly less than that of the new ones, but then not all required edges were being patrolled through the old routes. Note that the average, longest and shortest routes were taken over all five solutions generated by TabuGuard. Lastly, the degree to which the estate's road network is evenly patrolled can be

Table 4: Comparison between old Midfield-Estate routes and Tabu-Guard routes for three available guards.

	Old patrol routes	TabuGuard patrol routes
Average route length (m)	3342	3372
Longest route w_{max} (m)	3775	3381
Shortest route w_{min} (m)	2764	3359
$w_{max} - w_{min}$ (m)	1011	22

analysed by counting the number of times that each required edge is patrolled during a day. Each edge's traversal count for the old and new routes are shown in Figures 3 and 4. For the comparison we assumed that each of the five solutions of Tabu-Guard will be implemented twice during a day, and that the three old routes were patrolled ten times during a day. The difference between the most and least patrolled edges for the old routes was fifty traversals, while the difference for the Tabu-Guard routes is only ten. The analysis for all k solutions generated by Tabu-Guard is presented in Table 5. Due to space limitations we only show the number of traversals for the most and least patrolled edges, and not the complete traversal graphs.

As shown in the case study Tabu-Guard is capable of generating improved patrol routes for a security estate. However, the case study consists of only one test instance. To further evaluate Tabu-Guard's poten-

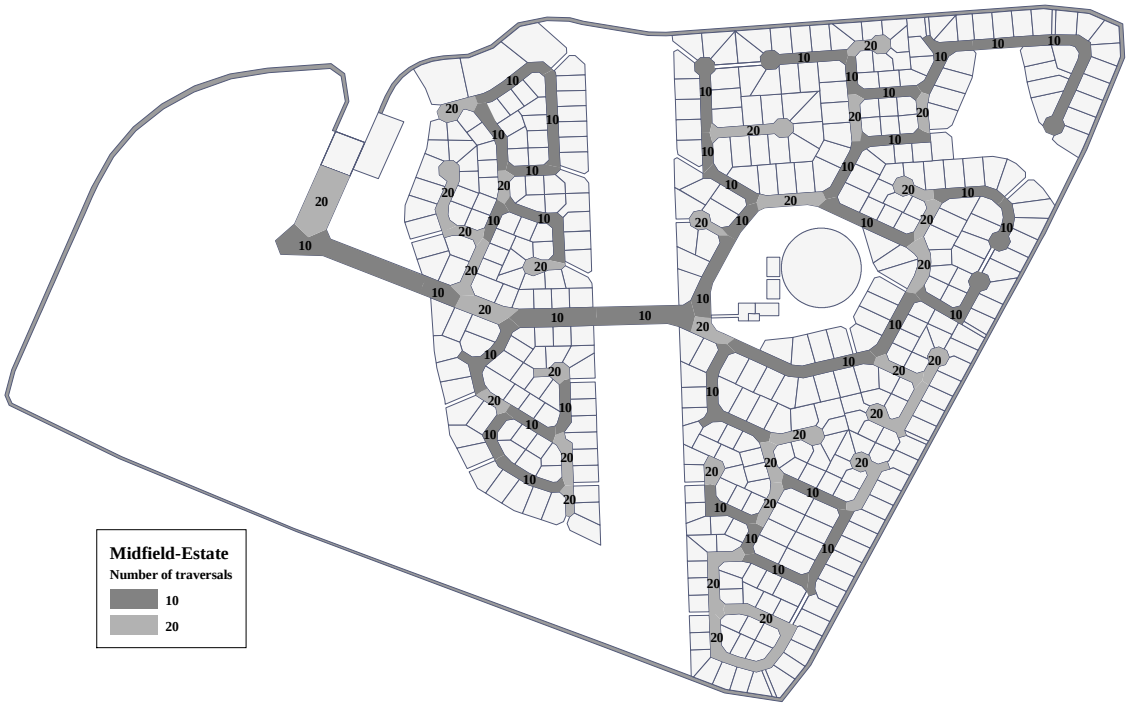

Figure 4: Number of street traversals through 24-hour patrolling for three guards with the new patrol routes.

Table 5: Results for the five Tabu-Guard solutions for each $k = 2, \ldots, 6$ chosen for implementation at Midfield-Estate.

k	Average route length (m)	Longest route (m)	Shortest route (m)	Difference (m)	Most traversals	Least traversals	Difference
2	4740	4780	4701	79	20	10	10
3	3372	3381	3359	22	20	10	10
4	2881	2897	2849	48	30	10	20
5	2588	2613	2527	86	30	10	20
6	2425	2447	2377	70	40	10	30

tial to solve MM k-CPP and MM k-RPP instances, we tested the algorithm on benchmark CARP instances from literature. The results are reported in the next section.

5. Further algorithm testing

For further evaluation of Tabu-Guard we tested the algorithm on networks of eight CARP instances from Li and Eglese (1996). The complete set contains 21 instances, but since the MM k-RPP and MM k-CPP do not model edge demand the problem set is reduced to eight instances with the others being duplicates. The eight instances used ranged in size from the smallest, containing 77 vertices, 51 required edges and 47 non-required edges, to the largest, having 140 vertices and 190 required edges. These ranges are consistent with network sizes of actual security estates that implement continuous guard patrolling. For two instances the complete edge set is required, i.e. $\boldsymbol{R} = \boldsymbol{E}$. The two instances are classified as MM k-CPP instances and the remaining six instances classified as MM k-RPP instances. To our knowledge this is the first work on MM k-RPP benchmark problems. The two MM k-CPP instances are solved by Ahr (2004) and Ahr and Reinelt (2006) with two tabu search algorithms called T10m and T-infinity (the latter is abbreviated T-INF in the remainder of this section), though the only difference between the algorithms is that a time limit of 10 minutes is imposed on T10m, after which the algorithm automatically terminates, whereas no time limit is imposed on T-INF.

For the benchmark problems Tabu-Guard generated and improved five initial solutions for each k, where we always considered the range $k = 2, \ldots, 10$ guards. The three Tabu-Search setups using RIN, EN and

RINEN, respectively, were then called to further improve each initial solution. Each setup used a single tabu tenure and the number of iterations without improvement was fixed at 500. RIN and RINEN were coupled with Simple-Aggressive-Tabu and executed using a tabu tenures of $\alpha = 6$ and $\alpha = 8$, respectively, whereas EN was coupled with Simple-Tabu and used a tabu tenure of $\alpha = 6$.

5.1. Results for MM k-CPP instances

Results for Tabu-Guard on the *egl-e4-A* instance, with a comparison to the solutions found by Ahr (2004) and Ahr and Reinelt (2006), are shown in Table 6. The table shows the w_{max} value of the best so-

Table 6: Computational results of three Tabu-Guard (TG) setups on the *egl-e4-A* instance of Li and Eglese (1996), with $|\boldsymbol{V}| = 77$, $|\boldsymbol{E}| = 98$ and $|\boldsymbol{R}| = 98$. The execution times reported are the total time taken to generate and improve five solutions for each k. The average w_{max} values are calculated in terms of lower bound gap values for each k.

		TG-RIN		TG-EN		TG-RINEN		T-10m	T-INF
k	LB	w_{max}	Time (s)	w_{max}	Time (s)	w_{max}	Time (s)	w_{max}	w_{max}
2	1685	1878	137.91	1810	1388.45	1828	886.25	1827	1816
3	1124	1311	178.54	1309	1020.21	1309	805.37	1352	1333
4	843	1112	229.12	1089	577.17	1090	613.76	1151	1102
5	820	966	235.97	951	446.63	956	523.34	1066	958
6	820	879	238.54	877	272.50	877	371.61	954	916
7	820	865	243.56	866	95.93	865	157.60	906	872
8	820	839	84.81	839	11.20	843	33.95	872	870
9	820	826	33.54	827	5.45	827	11.75	872	826
10	820	820	24.83	820	8.78	820	7.33	836	820
Average LB gap (%) and time (s)		*8.75%*	*156.31s*	*8.04%*	*425.15s*	*8.24%*	*379.00s*	*12.43%*	*9.30%*

lution found by each of the three Tabu-Guard algorithm setups over the five initial solutions. The reported computational times are then the total time of generating and improving all five initial solutions. Unfortunately, no execution times are given by Ahr (2004) and Ahr and Reinelt (2006) for T-10m and T-INF, though we can infer that the execution time of T-10m is always less than 600 seconds.

On average all three Tabu-Guard setups outperformed both T-10m and T-INF. Tabu-Guard found one optimal, one existing best and six new best solutions. Both Tabu-Guard-EN and Tabu-Guard-RINEN outperformed T-10m and T-INF for $k = 3, \ldots, 8$, whereas Tabu-Guard-RIN outperformed T-INF for $k = 3, 4, 6, 7, 8$ and T-10m for $k = 3, \ldots, 10$. In terms of solution quality Tabu-Guard-EN performed the best, followed closely by Tabu-Guard-RINEN and then Tabu-Guard-RIN. The latter dominates the other two setups in terms of execution time, where Tabu-Guard-EN, on average performs, the worst.

Results for Tabu-Guard on the *egl-s4-A* instance are presented in Table 7. Again, on average, all three

Table 7: Computational results of three Tabu-Guard (TG) setups on the *egl-s4-A* instance of Li and Eglese (1996), with $|\boldsymbol{V}| = 140$, $|\boldsymbol{E}| = 190$ and $|\boldsymbol{R}| = 190$. The execution times reported are the total time taken to generate and improve five solutions for each k. The average w_{max} values are calculated in terms of lower bound gap values for each k.

		TG-RIN		TG-EN		TG-RINEN		T-10m	T-INF
k	LB	w_{max}	Time (s)	w_{max}	Time (s)	w_{max}	Time (s)	w_{max}	w_{max}
2	2607	2761	775.89	2736	11451.37	2753	6349.09	2682	2651
3	1738	1894	1058.92	1874	7202.03	1875	6244.68	2053	1901
4	1304	1609	1309.34	1570	4516.17	1584	4103.79	1688	1552
5	1043	1379	1668.21	1322	3245.65	1315	3689.13	1470	1332
6	1027	1179	1519.03	1174	2485.17	1167	3357.80	1366	1241
7	1027	1107	1866.17	1107	2082.88	1101	3148.44	1255	1126
8	1027	1065	1630.70	1057	869.90	1056	1777.58	1208	1082
9	1027	1027	1404.71	1036	479.86	1027	665.54	1158	1053
10	1027	1027	400.07	1027	120.71	1027	153.45	1141	1050
Average LB gap (%) and time (s)		*8.98%*	*1292.56s*	*8.16%*	*3605.97s*	*8.05%*	*3276.61s*	*16.58%*	*9.30%*

Tabu-Guard setups outperformed both T-10m and T-INF. Tabu-Guard found two optimal and six new best

solutions. Tabu-Guard-EN and Tabu-Guard-RINEN outperformed T-10m and T-INF for $k = 3, 5, \ldots, 10$, whereas Tabu-Guard-RIN outperformed T-10m for all k values. There does seem to be inconsistencies between the results reported by Ahr (2004) and Ahr and Reinelt (2006) on instance *egl-s4-A*. Though both use the same algorithm, Ahr and Reinelt (2006) report the average LB gap of T-INF to be 11.88%, whereas results reported by Ahr (2004) and shown in Table 7 show the average to be 9.30%. Tabu-Guard-RINEN produced slightly better solutions than Tabu-Guard-EN, followed by Tabu-Guard-RIN. The computational times of Tabu-Guard were much higher for the *egl-s4-A* instance than with *egl-e4-A*. Tabu-Guard-EN took on average more than 3600 seconds to generate and improve five solutions, whereas the quickest setup, Tabu-Guard-RIN, took on average 1293 seconds. The execution times can be reduced by generating multiple initial solutions, and improving only the best initial solution; or by reducing the number of allowed moves without improvement.

5.2. Results for MM k-RPP instances

Summary results for Tabu-Guard on the six MM k-RPP instances adapted from Li and Eglese (1996) are shown in Table 8. We only report on the average results over $k = 2, \ldots, 10$; for the complete results

Table 8: Average computational results calculated over $k = 2, \ldots, 10$ of three Tabu-Guard (TG) setups on the MM k-RPP *egl* instances of Li and Eglese (1996). The average execution times reported are calculated over the total time taken to generate and improve five solutions for each k.

				TG-RIN		TG-EN		TG-RINEN	
File	***V***	***E***	***R***	LB Gap (%)	Time (s)	LB Gap (%)	Time (s)	LB Gap (%)	Time (s)
egl1-e1	77	98	47	2.51	14.90	2.11	44.90	2.49	43.01
egl1-e2	77	98	72	5.58	52.12	5.16	138.14	5.64	135.69
egl1-e3	77	98	87	7.95	115.37	7.46	263.13	8.02	278.71
egl1-s1	140	190	115	8.40	49.65	7.75	152.10	8.01	137.80
egl1-s2	140	190	147	13.31	560.16	11.61	1531.34	12.21	1570.17
egl1-s3	140	190	159	13.26	773.48	10.62	2169.72	11.12	2135.04

please contact the authors. Since these are the first reported results on MM k-RPP benchmark problems, analysis of the Tabu-Guard setups are limited to LB gaps. As with the previous tests, the Tabu-Guard-EN setup produced the best quality solutions, with a worst average LB gap of 11.61%. Again we observed large LB gaps for small k values, which can be partly attributed to the weakness of the bounds. The execution time of Tabu-Guard-EN and Tabu-Guard-RINEN were very similar, whereas Tabu-Guard-RIN is by far the quickest of the three setups. Furthermore, its LB values are within 3% of the other strategies, making the setup ideal when solutions have to be generated under time constraints. Based on the test results, when no time constraints are present, we recommend using Tabu-Guard-EN to solve the MM k-CPP and MM k-RPP.

6. Conclusion

This paper has demonstrated how the problem of designing patrol routes for security estates can be modelled as MM k-RPPs and MM k-CPPs. A tabu search algorithm capable of solving both problems was developed and tested on the road network of a a security estate, and solutions showed a significant improvement on the old patrol routes. Furthermore, the tabu search was used to generate a multitude of solutions, which resulted in the patrolling of the estate being unpredictable. As a final evaluation, our algorithm was tested on benchmark problems for the MM k-CPP. Results obtained show the algorithm to outperform the only existing algorithm for the MM k-CPP. We also tested our algorithm on new benchmark instances for the MM k-RPP. Results on both the MM k-CPP and MM k-RPP instances show that the algorithm is robust enough to generate quality patrol routes on different road networks.

There still exist opportunities to improve the patrolling of the estate even further. Most notable are the improvement of unpredictable patrolling and the placement of checkpoints. The strategy presented in this paper for unpredictable patrolling was to generate a multitude of solutions and then randomly choose and implement these solutions. Unfortunately, there were no measurement criteria in terms of the diversification of the chosen solutions. For unpredictable patrolling, solutions should be significantly dissimilar, while still being of high quality. This can be accomplished by finding local optima in the solution space that are

sufficiently dissimilar. A possible way to achieve this is by incorporating mechanisms into the tabu search algorithm that will guide the algorithm to these local optima.

As for the checkpoint placement, the solution in this paper requires all the checkpoints associated to a guard's route to be visited by the guard. A better solution would be to determine the minimum amount of checkpoints to be visited that will still lead to the patrol routes being traversed. Such an approach would be advantageous as the guards would then be able to concentrate on the actual patrolling of the estate instead of checkpoint visitations.

Acknowledgements

The authors would like to thank the management of Midfield-Estate for allowing us access to the estate's security services and for providing the actual data set. Lastly, we acknowledge the work of the anonymous referees for their valuable contribution to the improvement of the quality of the paper.

References

Ahr, D. (2004). *Contributions to multiple postmen problems*. PhD thesis, Ruprecht-Karls-Universität, Heidelberg.

Ahr, D. and Reinelt, G. (2002). New heuristics and lower bounds for the min-max k-Chinese postman problem. In Möhring, R. and Raman, R., editors, *Algorithms ESA 2002, 10th Annual European Symposium Rome, Italy, September 1721, 2002 Proceedings*, volume 33, pages 7–19. Heidelberg: Springer Berlin.

Ahr, D. and Reinelt, G. (2006). A tabu search algorithm for the min-max k-Chinese postman problem. *Comp Oper Res*, 33(12):3403–3422.

Amberg, A., Domschke, W., and Voß, S. (2000). Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *Eur J Oper Res*, 124(2):360–376.

Arkin, E. M., Hassin, R., and Levin, A. (2006). Approximations for minimum and min-max vehicle routing problems. *J Algorithms*, 59(1):1–18.

Benevant, E., Corberán, A., Plana, I., and Sanchis, J. M. (2009). Min-max k-vehicles windy rural postman problem. *Netw*, 54(4):216–226.

Brandão, J. and Eglese, R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. *Comp Oper Res*, 35(4):1112–1126.

Corberán, A. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Netw*, 56(1):50–69.

Dror, M., editor (2000). *Arc Routing: Theory, Solutions, and Applications*. Boston: Kluwer Academic Publishers.

Edmonds, J. and Johnson, E. L. (1973). Matching, Euler tours and the Chinese postman. *Math Program*, 5(1):88–124.

Eiselt, H. A., Gendreau, M., and Laporte, G. (1995a). Arc routing problems, part I: The Chinese postman problem. *Oper Res*, 43(2):231–242.

Eiselt, H. A., Gendreau, M., and Laporte, G. (1995b). Arc routing problems, part II: The rural postman problem. *Oper Res*, 43(3):399–414.

Frederickson, G., Hecht, M., and Kim, C. (1978). Approximation algorithms for some routing problems. *SIAM J Comput*, 7(2):178–193.

Glover, F. (1989). Tabu search – part I. *ORSA J Comput*, 1(3):190–206.

Glover, F. (1990). Tabu search – part II. *ORSA J Comput*, 2(1):4–32.

Glover, F. W. and Laguna, M. (1998). *Tabu Search*. Springer.

Golden, B. L. and Wong, R. T. (1981). Capacitated arc routing problems. *Netw*, 11(3):305–315.

Greistorfer, P. (2003). A tabu scatter search metaheuristic for the arc routing problem. *Comp Ind Eng*, 44(2):249–266.

Hertz, A., Laporte, G., and Mittaz, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Oper Res*, 48(1):129–135.

Lacomme, P., Prins, C., and Ramdane-Chérif, W. (2004). Competitive memetic algorithms for arc routing problems. *Ann Oper Res*, 131(4):159–185.

Lenstra, J. and Rinnooy Kan, A. (1976). On general routing problems. *Netw*, 6(3):273–280.

Li, L. Y. O. and Eglese, R. W. (1996). An interactive algorithm for vehicle routeing for winter - gritting. *J Oper Res Soc*, 47(2):2.

Wøhlk, S. (2008). A decade of capacitated arc routing. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 29–48. Springer US.

Wolfler Calvo, R. and Cordone, R. (2003). A heuristic approach to the overnight security service problem. *Comp Oper Res*, 30(9):1269–1287.

A. Tabu-Guard algorithms

Algorithm procedures.

Algorithm 6: Insert-Arc

Input : Route $\boldsymbol{C}_i$ and arc u.
Output: Route $\boldsymbol{C}_i$ with arc u assigned to the route.

Let n be the number of arcs, including dummy arcs, serviced in route $\boldsymbol{C}_i$;
foreach $v \in (u, inv(u))$ **do**
  for $j \leftarrow 2$ **to** n **do**
    $\boldsymbol{C}'_i \leftarrow \boldsymbol{C}_i$;
    Insert v in route $\boldsymbol{C}'_i$ in position j;
    $\Delta \boldsymbol{S}_j \leftarrow w(\boldsymbol{C}_i) - w(\boldsymbol{C}'_i)$;
  Find $t \leftarrow \arg\min\{\Delta \boldsymbol{S}_k : k \in (2, \ldots, n)\}$;
  Let $\boldsymbol{C}_i^{(v)} \leftarrow \boldsymbol{C}_i$ and insert v in $\boldsymbol{C}_i^{(v)}$ in position t;
Find $\boldsymbol{C}_i^{(l)}$ where $l \leftarrow \arg\min\{w(\boldsymbol{C}_i^{(v)}) : v \in (u, inv(u))\}$;
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}_i^{(l)}$;
return $(\boldsymbol{C}_i)$

With Algorithm 7, $\boldsymbol{C}_i(l)$ is defined as the arc in position l in route $\boldsymbol{C}_i$.

Algorithm 7: EXCHANGE-ARCS

Input : Route $\boldsymbol{C}_i$.
Output: Potentially improved route $\boldsymbol{C}_i$.
Let n be the number of arcs, including dummy arcs, serviced in route $\boldsymbol{C}_i$;
Let $t \leftarrow 0$;
for $l \leftarrow 2$ **to** $n-2$ **do**
  for $m \leftarrow i+1$ **to** $n-1$ **do**
    $t \leftarrow t+1$;
    $\boldsymbol{C}^{(1)}_{temp}, \boldsymbol{C}^{(2)}_{temp}, \boldsymbol{C}^{(3)}_{temp}, \boldsymbol{C}^{(4)}_{temp}, \leftarrow \boldsymbol{C}_i$;
    Let $\boldsymbol{C}^{(1)}_{temp}(l) \leftarrow \boldsymbol{C}_i(m)$ and $\boldsymbol{C}^{(1)}_{temp}(m) \leftarrow \boldsymbol{C}_i(l)$;
    Let $\boldsymbol{C}^{(2)}_{temp}(l) \leftarrow \boldsymbol{C}_i(m)$ and $\boldsymbol{C}^{(2)}_{temp}(m) \leftarrow inv(\boldsymbol{C}_i(l))$;
    Let $\boldsymbol{C}^{(3)}_{temp}(l) \leftarrow inv(\boldsymbol{C}_i(m))$ and $\boldsymbol{C}^{(3)}_{temp}(m) \leftarrow \boldsymbol{C}_i(l)$;
    Let $\boldsymbol{C}^{(4)}_{temp}(l) \leftarrow inv(\boldsymbol{C}_i(m))$ and $\boldsymbol{C}^{(4)}_{temp}(m) \leftarrow inv(\boldsymbol{C}_i(l))$;
    Find $\boldsymbol{C}^{(k)}_{temp}$ where $k \leftarrow \arg\min\{\boldsymbol{C}^{(l)}_{temp} : q \in (1,\ldots,4)\}$;
    $\boldsymbol{C}'_t \leftarrow \boldsymbol{C}^{(k)}_{temp}$;
Find $\boldsymbol{C}'_k$ where $k \leftarrow \arg\min\{w(\boldsymbol{C}'_q) : q \in (1,\ldots,q)\}$;
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}'_k$;
return $(\boldsymbol{C}_i)$

Algorithm 8: REMOVE-INSERT-ARCS

Input : Route $\boldsymbol{C}_i$.
Output: Potentially improved route $\boldsymbol{C}_i$.
foreach $u \in \boldsymbol{C}_i$ **do**
  Let $\boldsymbol{C}'_i \leftarrow \boldsymbol{C}_i$ and remove u from $\boldsymbol{C}'_i$;
  $\boldsymbol{C}^{(u)}_{temp} \leftarrow$ INSERT-ARC$(\boldsymbol{C}'_i, u)$;
Find $\boldsymbol{C}^{(k)}_{temp}$ where $k \leftarrow \arg\min\{w(\boldsymbol{C}^{(u)}_{temp}) : u \in \boldsymbol{C}_i\}$;
$\boldsymbol{C}_i \leftarrow \boldsymbol{C}^{(k)}_{temp}$;
return $(\boldsymbol{C}_i)$