Computers & Operations Research 37 (2010) 1870–1876

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Solving the mobile mapping van problem: A hybrid metaheuristic for capacitated arc routing with soft time windows

Pieter Vansteenwegen[a], Wouter Souffriau[a,b], Kenneth Sörensen[c,*]

[a]Centre for Industrial Management, Katholieke Universiteit Leuven, Celestijnenlaan 300A, 3001 Leuven, Heverlee, Belgium
[b]Information Technology, KaHo Sint-Lieven Gebroeders, Desmetstraat 1, 9000 Gent, Belgium
[c]Universiteit Antwerpen, Faculteit Toegepaste Economische, Wetenschappen, Prinsstraat 13, 2000 Antwerp, Belgium

ARTICLE INFO

Available online 18 May 2009

Keywords:
Vehicle routing
Arc routing
Mobile mapping van
Tele Atlas
Soft time windows

ABSTRACT

Creating digital maps often requires driving around the streets in a so-called "mapping van". Tele Atlas, the world leading supplier of maps and map data, uses a fleet of such vehicles to take pictures of streets and road signs. Minimising the number of days that a vehicle needs to traverse all streets in a given region gives rise to a capacitated arc routing problem. A specific characteristic of this problem, however, is the fact that taking pictures in the direction of the sun should be avoided as much as possible. This requirement adds soft time windows to the problem.

In this paper we solve the mobile mapping van problem by transforming it into a vehicle routing problem with soft time windows. We then apply a hybrid metaheuristic, consisting of a local search phase to decrease the number of days and an iterated local search phase to minimise the time window violations. An exact linear programming solver is embedded to calculate the ideal starting time of the vehicle in each street.

Our method is tested on academic and real-life problem instances and shown to be effective.

1. Introduction

Tele Atlas is the world leading supplier of geographical data. To update and improve its data, "mobile mapping vans" (see Fig. 1) drive around to take pictures of streets and road signs. Afterwards, these pictures are analysed and the information collected is used to update the map database. In order to save costs, Tele Atlas aims to minimise the number of days required to travel through all the streets in a certain region, using a single van. Determining the order in which to drive through each street in a certain region is the objective of this paper.

Tele Atlas distinguishes three different street types, that have to be traversed in different ways. In a one way street, pictures can only be taken while driving in the driving direction. In a relatively narrow two way street, pictures can be taken in both directions at the same time. To cover a broad two way street—in which the two driving directions are separated by a berm for example—the van has to drive through the street in both directions. Such broad two way streets can be modelled as two one way streets.

The problem to minimise the total time required to travel through all streets can be modelled as a mixed capacitated arc routing problem (MCARP), in which the roads in the network can either be undirected edges (narrow two way streets) or directed arcs (one way streets or broad two way streets that have been modelled as two one way streets).

Besides minimising the number of days, Tele Atlas wants to minimise the number of pictures taken towards the sun, since many of these pictures are useless. As a result, many streets can only be successfully traversed during a certain time of the day. This extra objective can be modelled with soft time windows, based on the orientation of the streets. Taking pictures towards the sun, i.e. violating the time window, will be penalised in the objective function. Note that Tele Atlas does not require vans to completely avoid driving towards the sun. If that was the case, this problem should be modelled with hard time windows.

In order to solve the MCARP with soft time windows efficiently, the problem is first converted to a vehicle routing problem with soft time windows (VRPSTW). This technique is based on a technique, described in the literature [1,2,13,17], to convert an undirected CARP into a VRP, and which has proven to work well before.

The remainder of this paper is organised as follows. In the next section, a short literature review is presented. In Section 3, the problem formulation and the transformation to a VRPSTW is discussed.

* Corresponding author. Tel.: +32 3 220 40 48; fax: +32 3 220 49 01.
E-mail addresses: pieter.vansteenwegen@cib.kuleuven.be (P. Vansteenwegen), wouter.souffriau@kahosl.be (W. Souffriau), kenneth.sorensen@ua.ac.be (K. Sörensen).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.05.006

Fig. 1. A Tele Atlas "mapping van".

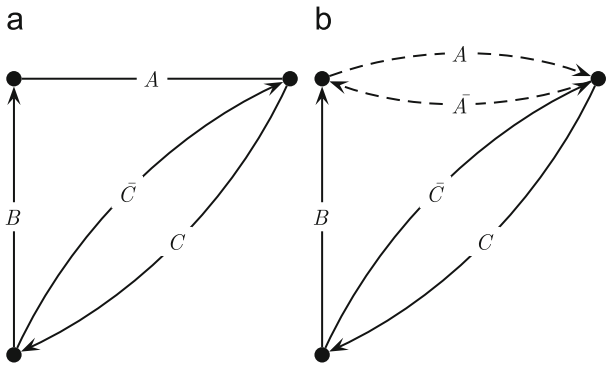


Fig. 2. An edge (A) is transformed into two arcs (A and $\bar{A}$), of which only one needs to be traversed. (a) Example road network with undirected edges and directed arcs. (b) Example road network with edges replaced by arcs.

Section 4 describes the solution procedure and in Section 5 experimental results are discussed. Section 6 concludes this paper and adds suggestions for further research.

2. Literature review

The problem discussed in this paper is a capacitated arc routing problem with soft time windows. Arc routing has recently received a large amount of research interest, but arc routing with time windows is still a relatively underexplored subject, although some metaheuristic optimisation approaches exist. Aminu and Eglese [1] tackle the Chinese postman problem with time windows using a constraint programming approach. They develop two approaches, one of which tackles the problem directly, whereas the other transforms the problem into a node routing problem. Brandáo and Eglese [4] develop a tabu search heuristic for the capacitated arc routing problem. Labadi et al. [12] develop a GRASP method with path relinking for the capacitated arc routing problem with time windows. In Belenguer et al. [3], a memetic algorithm is designed to tackle the MCARP and this is currently the only published metaheuristic for the MCARP.

Contrary to arc routing, time windows have been an important object of study in the area of vehicle routing. Undoubtedly the best-studied problem in this area is the vehicle routing problem with (hard) time windows (VRPTW), for which a large number of efficient heuristic methods exists [5,6]. The literature on vehicle routing with soft time windows, on the other hand, in which it is allowed to arrive early or late at a customer's location thereby incurring a penalty cost, is much more limited. An early approach to model and solve a multi-objective VRPSTW can be found in Min [15], who uses this problem to model the case of a public library distribution system. More recent approaches can be found in Taillard et al. [21], who develop a tabu search heuristic and Ioannou et al. [11], who develop a heuristic based on a nearest-neighbour approach. In Calvete et al. [7] a goal programming method is developed. Min [15] developed a multi-objective method to solve a public library distribution system problem. One possible way to solve an arc routing problem, also used in this paper, is to transform it into an equivalent node routing problem. The first transformation appeared in Pearn et al. [17], a more recent and more efficient one can be found in Longo et al. [13].

3. Problem formulation and transformation

We are given a single mapping van with a fixed maximum travel time per day. The mapping van starts at the same node (corresponding to a hotel) each day and has to return to this hotel at the end of the day, before exceeding its maximum travel time for that day. Every (directed) arc or (undirected) edge in the graph has a given, fixed, travel time, as well as a soft time window, i.e. an earliest time at which traversal of this arc or edge should begin and a latest time traversal should end if no penalty cost is to be incurred.

The penalty cost for not (entirely) traversing an arc during its time window is based on the time window violation. A time window violation is defined as the time difference between the start of the visit and the opening of the time window or between the closing of the time window and the end of the visit, divided by the width of the corresponding time window.

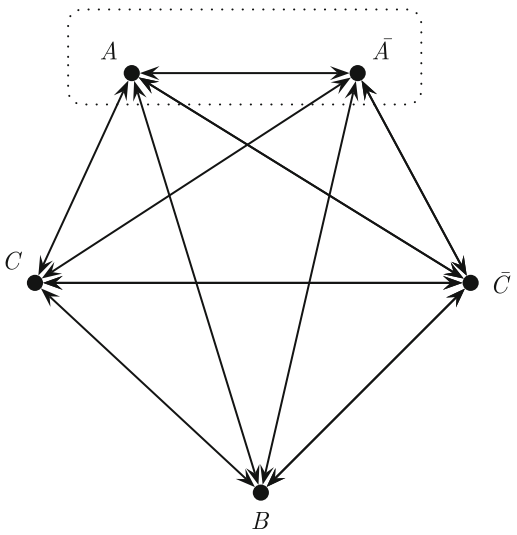
The total travel time in any given day is the sum of the travel times of all streets traversed that day. On the one hand, the travel time per day is constrained by a fixed maximum, on the other hand, the objective is to reduce the travel time per day in order to make it possible to reduce the required number of days. In general, some streets will have to be traversed several times in order to completely traverse the map. Of course, pictures have to be taken only once in any given street. When traversing a street several times, the best time (i.e. the one that incurs the lowest penalty) can be chosen to take pictures.

The objective of this capacitated arc routing problem with (soft) time windows is to determine the order in which to visit all streets, i.e. travel all arcs and edges of the map, in such a way that the weighted sum of (a) the number of days and (b) the total time window violation, is minimised.

As mentioned, our mixed capacitated arc routing problem with soft time windows is first transformed to a vehicle routing problem with soft time windows. This conversion is done in two steps. In a first step, all (undirected) edges are replaced with two (directed) arcs of identical length, of which only one needs to be traversed (see Fig. 2 for an example). In a second step the arc routing problem is transformed into a vehicle routing problem by replacing each arc with a node (customer) to visit. This is shown in Fig. 3, in which the road network of Fig. 2 is transformed. The distance between each pair of nodes is determined as the shortest distance between the end node of the arc corresponding to the first customer and the starting node of the arc corresponding to the second customer, resulting in an asymmetric VRPSTW. This shortest distance is defined as the shortest path between the two nodes and can be calculated by any algorithm for the shortest path.

In the remainder of this paper, we will only use the transformed VRPSTW formulation of the mobile mapping problem and use the term "node" to refer to "the traversing of a street by a mobile mapping van".

a



b

from/to	A	$\bar{A}$	B	C	$\bar{C}$
A	–	0	$\|C\|$	0	$\|C\|$
$\bar{A}$	0	–	$\|A\|+\|C\|$	$\|A\|$	$\|A\|+\|C\|$
B	0	$\|A\|$	–	$\|A\|$	$\|A\|+\|C\|$
C	$\min(\|B\|,\|\bar{C}\|+\|\bar{A}\|)$	$\min(\|\bar{C}\|,\|B\|+\|A\|)$	0	–	0
$\bar{C}$	$\min(\|\bar{A}\|,\|C\|+\|B\|)$	0	$\|C\|$	0	–

Fig. 3. Transformation into a directed VRPSTW. (a) Transformed (directed) VRPSTW network, only one of nodes A and $\bar{A}$ needs to be visited. (b) Asymmetric distance matrix of the transformed problem.

4. Solution procedure

4.1. Framework

As mentioned, the objective function of our problem is a weighted sum of the number of required days and the total time window violations. The relative weights of each of these two criteria allow Tele Atlas to determine the relative importance of each criterion. Our approach uses several neighbourhood structures to improve the solution one step at a time. At each iteration, the best move is chosen by evaluating the objective function for all solutions in the neighbourhood of the current solution, potentially inducing a large computational overhead. We have therefore implemented two strategies to reduce the computing time of our algorithm.

As a first time-reducing strategy, an initial solution of the VRPSTW is improved in two phases. In the first phase, the number of days to visit all the nodes is minimised, not taking the time windows into account. In the second phase, while keeping the number of days constant, the total time window violation is reduced. Then, adding one or more days to the solution is considered if this significantly reduces the time window violations.

The total travel time of a certain solution is the sum of the costs of all the arcs included in this solution. Because of this, the travel time improvement or deterioration of a solution caused by a move only depend on the arcs that are either removed or added by this move. The travel time of a solution after a local move can therefore be computed from the cost of the initial solution by subtracting the cost of the removed arcs and adding the cost of the added arcs. For each move type this can be done in constant time, irrespective of the problem size. In contrast, to evaluate the change in time window violations after a certain move, the possible time window violation needs to be computed again for each node of the day under consideration, inducing a significant computational overhead. The total computation time of the algorithm would increase drastically if this expensive evaluation would also be required when moves are considered to reduce the number of days. As a second time-reducing strategy, we therefore use a shortcut procedure to quickly evaluate the minimum time window violation reduction of each potential move, and only calculate the best possible time window violation reduction when a selected move is executed (see Section 4.4.1).

Algorithm 1 in Section 4.5 presents an overview of the structure of the whole algorithm. First, all steps are explained one by one in the next sections.

4.2. Initialisation

The generation of an initial solution is based on the sequential insertion heuristic "I1" of Solomon [20]. A day is initialised with a visit to the geographically farthest unvisited node from the depot. Then, the remaining unvisited nodes are added based on two criteria. First, the best insertion position with respect to extra travel time is determined for each unvisited node. Secondly, the node with the highest value of "travel time from the depot" minus "extra travel time required for insertion" is selected for insertion. If the current day is full, a new day is initialised with the farthest unvisited node and unvisited nodes are added until all nodes are visited. During the initialisation phase, no time windows are taken into account.

4.3. Minimise the number of days

The first attempt to reduce the travel time, focuses on the edges that were converted into two corresponding nodes, of which only

one needs to be visited (see Fig. 3). The neighbourhood operator Turn will check for each included node if travel time can be saved by replacing it by its corresponding non-included node. This is the same as switching the driving direction through the edge.

Then, three neighbourhoods are explored in order to minimise the travel time in each day:

- Re-Opt: relocates a chain of consecutive nodes inside a day or from one day to another. In this way, Re-Opt generalises "Or-Opt" [16], relocating a chain inside a day, and "relocate" [9], moving a node from one day to another.
- 2-Opt* [18]: two days are combined so that the last nodes of one day are scheduled after the first nodes of the other day. The orientation of the routes is preserved.
- Exchange [19]: exchanges two nodes between different days.

These neighbourhoods are sequentially explored in a best improving fashion. If a local optimum is reached, Turn is applied again. This means that Turn is applied twice for each number of days; before and after all local search moves.

Next, a procedure called Reduce is applied that tries to move each node of a given day to one of the other days. If this appears feasible, the given day is removed, the travel time in the remaining days is minimised, again using the above mentioned neighbourhoods, and Reduce is applied again.

Only straightforward heuristics are implemented since not too much computation time should be spent to find the absolute minimal number of days to visit all nodes (not taking the time windows into account). For many instances, one or more days will be added anyway to this minimum in order to reduce the time window violations (and the overall objective) in the next step.

4.4. Minimise the time window violations

In order to minimise the time window violations, while keeping the number of days constant, iterated local search (ILS) [14] is used. ILS is a metaheuristic based on a simple idea: iteratively build sequences of solutions that are generated by an embedded local search heuristic. This leads to much better solutions than repeating random trials of the same heuristic. ILS has shown before to work well on problems with time windows [5,9,10].

4.4.1. Local search improvement

Starting from a solution with a given number of days and a minimised travel time per day, the number of time window violations is minimised without changing the number of days. As a local search heuristic in the ILS framework, Re-Opt is applied again. In this case, however, a move is only considered if the time window violations can be reduced.

Once the sequence of visiting the streets in a given day is fixed, the total time window violation can be further reduced by allowing the vehicle to strategically distribute the time it has still available in any given day, so that it waits before traversing some streets. In order to reduce time window violations in limited computation time, we use a fast heuristic to evaluate the minimal reduction in time window violation of each possible Re-Opt move, and a (slower) exact procedure to optimally redistribute the waiting time of the vehicle once a certain move has been chosen and the sequence of streets is fixed.

The heuristic procedure evaluates all potential moves from a given solution and determines in a relative fast way what the *minimal* violation reduction of that move will be. The move with the highest minimal (and positive) reduction is selected for execution.

Two different situations are distinguished when evaluating the minimal violation reduction: the relocation of a node within a day or the relocation of a node from one day to another.

When a node X is moved from day i to day j, all nodes that were scheduled after X in day i can be shifted forward (towards the beginning of the day) one by one. As long as the time window violation does not increase, these forward shifts are carried out and the possible violation reduction is calculated. When a shift would increase the total violation, the shift is not carried out and the rest of the nodes, and their violations, can remain unchanged. In day j, the nodes scheduled after the new position of node X, will be shifted backwards (towards the end of the day) due to the insertion of X. This can change the total time window violation. However, a waiting time might have been scheduled before some of these nodes. This waiting time will reduce the backward shift for all nodes scheduled later that day. As soon as the backward shift is reduced to zero, the remaining nodes (and their violations) can remain unchanged. Nodes scheduled before node X in day i or j, do not have to be moved and are, therefore, not considered when calculating the minimal reduction in the time window violation.

When a node X is relocated within a day, all nodes scheduled after its new position need to be considered for a backward shift and a change in violation. Similar to the previous case with a backward shift, scheduled waiting times can reduce the shift to zero. nodes scheduled before the new position of X do not have to be moved and are, again, not considered when calculating the minimal reduction in the time window violation. Only when node X is relocated towards the end of the day, the nodes between its previous and new position need to be shifted towards the beginning of the day. These shifts are required to make sure that node X can be inserted without exceeding the time budget. These shifts will also change the time window violation.

Finally, the change in time window violation of node X is taken into account in all cases. Some opportunities for reducing the time window violations are not considered in this quick evaluation. Only the minimal guaranteed reduction is calculated.

When the best move is chosen and executed, a straightforward linear programming problem is solved to obtain, for the new sequence of nodes, the minimal time window violations possible by distributing the excess time. In Fig. 4, $begin_i$ is the time the vehicle starts taking pictures in the street corresponding to node i, which may be later than the time it arrives (the vehicle may wait for the sun to change position and hence the time window violation to become less severe). $duration_i$ is the time it takes for the vehicle to traverse street i taking pictures and $distance(i,j)$ is the travel time between nodes i and j. $open_i$ and $close_i$ represent the earliest time the mapping van can start taking pictures and the latest time it has to finish without incurring any penalty cost. $overtime_i$ and $undertime_i$ represent the amount of time the van spends taking pictures after and before the time window limits, respectively.

This LP determines the optimal starting time ($begin_i \in \mathbb{R}^+$) for each node without changing the order of the nodes in the sequence, potentially allowing the van to wait before driving through a street, and is solved by an embedded linear programming solver. The reduction in violations will always be at least as high as the minimal reduction guaranteed by the heuristic evaluation, and often even higher.

4.4.2. Perturbation

During ILS, if a local optimum is reached, the solution will be shaken in order to reach another, and hopefully better, local optimum. During this perturbation phase, a number of nodes will be removed in each day. The operator Perturb uses two parameters: *NumberToRemove* indicates how many consecutive nodes to remove in each day, while *StartPosition* indicates the position in each day to

$$\min\ violations = \sum_{i=1}^{n}\left[\frac{overtime_i}{close_i - open_i} + \frac{undertime_i}{close_i - open_i}\right]$$

$$s.t.$$

$$begin_i \geq begin_{i-1} + duration_{i-1} + distance(i-1,i) \qquad i = 1,\ldots,n+1$$

$$begin_i \geq open_i - undertime_i \qquad i = 0,\ldots,n+1$$

$$begin_i + duration_i \leq close_i + overtime_i \qquad i = 0,\ldots,n$$

$$begin_0 \geq 0,\ begin_{n+1} = close_{n+1}$$

Fig. 4. LP to minimise the overall time window violations for a given sequence of nodes.

start the removal process. If, during the removal, the end location is reached, the removal continues after the start location. After the removal, all nodes following the removed nodes are shifted to the beginning of the day as much as possible, in order to maximise the opportunity for other nodes to be included. Next, all the removed nodes are inserted again based on the criteria used during the initialisation phase. If it is impossible to insert all nodes in this way, Perturb is first reversed, i.e. the removed nodes are inserted again in their original sequence, and then applied again after updating the parameters (*NumberToRemove*, *StartPosition*). How these parameters are updated is explained in Section 4.4.4. Again, no time windows are taken into account during insertion.

4.4.3. Acceptance criterion

As an acceptance criterion, "random walk" is used [14]. The heuristic always continues the search from the current solution, it never returns to the current best solution to continue. Of course, the best found solution is always retained. This random walk acceptance criterion enhances the diversification.

4.4.4. Iterated local search parameters

The ILS heuristic follows a loop until, during a fixed number of times, no improvements are identified for the best solution determined so far. After the perturbation phase, Re-Opt (taking into account time windows) is applied until a local optimum is reached. If this solution is better than the incumbent solution, the solution is recorded and *NumberToRemove* is reset to one for the next perturbation phase. After each perturbation phase, *StartPosition* is increased by the value *NumberToRemove* and *NumberToRemove* is increased by one for the next perturbation phase. When *NumberToRemove* is greater than or equal to $n/(3 \times m)$, it is reset to 1 (n is the number of nodes; m is the number of days). By using the perturbation parameters as described above, other nodes are removed during every perturbation phase and during the entire procedure, most likely, every node is removed at least once. This should ensure a better exploration of the solution space.

The maximum number of nodes to remove ($n/(3 \times m)$) and the maximum number of iterations without improvement (n/m) are the only parameters to predetermine in this heuristic. Preliminary testing has showed that further increasing the maximum number of iterations without improvement does not significantly improve the results and only causes longer computation times. The difficulty of the problem, and therefore the required number of iterations, depends on the average nodes per day ($=n/m$).

After minimising the time window violations for this fixed number of days, an extra day is added to the solution in order to increase the possibility to further reduce the time window violations. Again the ILS heuristic is applied to minimise the violations for this number of days. If the overall objective, i.e. the weighted sum of the number of days and time window violations, is reduced by adding one day to the initial solution, another day is added and the problem is solved again.

4.5. Algorithm overview

An overview of the hybrid metaheuristic solution procedure for the mobile mapping van problem can be found in Algorithm 1.

Algorithm 1. Pseudo-code of the hybrid metaheuristic

```
Initialisation;
Minimise number of days:
while Reduce is possible do
    Turn;
    Re-Opt;
    2-Opt*;
    Exchange;
    Turn;
Minimise violations:
while adding an extra day decreases the objective do
    Re-Opt + LP;
    while NumberOfNoImprovements < n/m do
        Update : NumberToRemove & StartPosition;
        Perturb (NumberToRemove & StartPosition);
        Insert;
        Re-Opt + LP;
    Add an extra day;
```

5. Experimental results

The ILS heuristic was tested on the well-known test set of Solomon ($n = 100$) and eight real-life planning problems from a Flemish bicycle network. The overall objective during the experimental results equals "the required number of days" plus "the time window violations divided by 10". All computations were executed on a personal computer Intel Core 2 Duo with 2.53 GHz processor and 3.45 GB RAM.

Table 1 presents the results for the test set of Solomon. Column one gives the instance name. As a reference, column two states the standard number of days used to solve the regular VRP with hard time windows and column three states the objective value of the initial solution (before minimising the violations). In column four, five and six the number of days, the time window violation and the objective function of the solution are presented. The computation time in the last column is expressed in seconds. On average, the computation time for these problems is 7 s and the time window violations are reduced with 32% compared to the initial violations. Except for instance r104, the number of days in the final solution is equal to or lower than the standard required number of days for the instances with hard time windows. These problems were also solved by a commercial MIP solver (CPLEX 11.0). Starting from the heuristic solution, none of these solutions could be improved, when a time limit of 2 h per instance is imposed.

CARPSTW instances are generated from a well-known road network for biking in East Flanders, Belgium, consisting of 893 landmarks with 1289 edges and 91 arcs in between, with a total length of

3328 km. The entire network is split up into different regions, Meetjesland, Scheldeland, Vlaamse Ardennen and Waasland, which result in four test instances. Adjacent regions are combined into larger instances, resulting in four more test instances. Table 2 presents the characteristics of the different instances.

The real-life arc routing instances are first converted to vehicle routing instances. All pairwise shortest distances are calculated by the Floyd–Warshall algorithm [8]. The distance matrix is divided by the travel velocity, fixed at 60 km/h. The visiting time of each point is calculated by dividing the length of the corresponding arc by the so-called picture velocity, fixed at 10 km/h. In order to calculate the time window for each point, we assume that the sun rises in the east at 8 AM and sets in the west at 8 PM. The sun moves 180° clockwise in 720 min, resulting in a speed of 0.25°/min. The orientation of an arc is calculated by $90 + \arctan(\Delta x/\Delta y)$. Dividing this orientation by the speed of the sun results in midsuntime, the time the sun is in line with the arc. If this time is situated before noon (2 PM), we opt to visit the arc in the afternoon, and vice versa. This paper assumes that the sun prevents taking high quality pictures from *midsuntime* − 120 until *midsuntime* + 120, i.e. 2 h before and 2 h after the sun is in line with the street the pictures are being taken of. In case of an afternoon visit, the open of the point is set to *midsuntime* + 120 and the close to the close of the endpoint. In case of a visit before noon, the open of the point is set to the open of the start point and the close to *midsuntime* − 120. Each day starts in the morning at 8 AM and ends in the evening at 6 PM.

Table 3 presents the results for these real-life problems. Column one gives the instances name. Column two states the objective value of the initial solution (before minimising the violations). In column three and four the number of days and the time window violation of the solution are presented. Because a higher total violation can be expected when more points need to be visited, column five gives the violation per point. Column six states the objective value of the heuristic solution and column seven compares this value with the initial objective function. The computation time in the last column is expressed in seconds. Since (very) large instances are solved in a few hours of calculation, the computation time for these problems can be considered acceptable. The time window violations are reduced with 40% compared to the initial violations and the average violation per point is 0.50. This may still seem high, but it is mainly a result of our policy to implement only one time window per street and will surely improve when multiple time windows per street are allowed.

Finally, we have also attempted to compare our algorithm to another approach from the literature. In Taillard et al. [21], an algorithm for vehicle routing with soft time windows is used to solve Solomon's problems with hard time windows (using a high penalty for violations). We have applied our algorithm thereby setting the weight of the time window violations criterion in the objective function to a very high value (100 instead of 0.1). Unfortunately, there are considerable differences in the problem formulation, more specifically the fact that in [21] a vehicle is not allowed to visit a certain customer before its time window opens (the vehicle has to wait until the time window opens), whereas in our approach the vehicle may start the visit early, incurring a penalty. Since the requirement to wait before starting a service simplifies the problem formulation, our approach did not yield comparable results or even results that could be used for further analysis. Furthermore, it is no surprise that our approach for soft time windows is not suited to deal with problems with hard time windows. We have therefore decided not to elaborate on the details of this experiment.

Table 1
Results on test instances from Solomon [20] ($n = 100$).

Name	VRPTW # days	Initial objective	# Days	Violation	Objective	CPU (s)
c101	10	24.5	10	0.0	10.0	6
c102	10	21.6	9	5.6	9.6	4
c103	10	19.0	9	4.8	9.5	5
c104	10	15.9	9	0.1	9.0	2
c105	10	18.9	10	0.0	10.0	6
c106	10	19.0	9	8.1	9.8	7
c107	10	15.9	9	4.7	9.5	4
c108	10	13.8	9	4.4	9.4	8
c109	10	11.5	9	0.1	9.0	4
r101	19	36.7	13	28.1	15.8	16
r102	17	27.4	13	16.9	14.7	12
r103	13	22.6	12	15.9	13.6	5
r104	9	16.8	10	7.9	10.8	5
r105	14	20.4	11	16.8	12.7	11
r106	12	16.4	11	12.3	12.2	8
r107	10	14.5	10	8.7	10.9	4
r108	9	12.3	9	5.0	9.5	3
r109	11	13.6	9	18.0	10.8	8
r110	10	12.1	9	13.1	10.3	6
r111	10	13.2	10	8.9	10.9	4
r112	9	9.9	9	3.9	9.4	2
rc101	14	21.4	12	9.4	12.9	14
rc102	12	17.4	11	9.3	11.9	9
rc103	11	14.7	10	8.8	10.9	5
rc104	10	11.5	9	10.5	10.0	2
rc105	13	19.6	12	9.8	13.0	10
rc106	11	14.0	10	5.0	10.5	8
rc107	11	12.0	9	7.8	9.8	6
rc108	10	10.3	9	3.6	9.4	5
Average		17.1		8.5	10.9	6.5

Table 2
Characteristics of the real-life test instances.

Name	Landmarks	# Edges	# Arcs	Length (km)
Meetjesland	93	124	1	252
Waasland	136	186	13	563
Vlaamse Ardennen	149	207	7	474
Meetjesland + Waasland	343	487	21	1230
Scheldeland	366	500	63	1412
Scheldeland + Waasland	472	655	77	1887
Scheldeland + Vlaamse Ardennen	672	956	77	2469
Entire network	891	1289	91	3328

Table 3
Results on real-life instances.

Name	Initial objective	# Days	Violation	Violation/point	Objective	Improvement (%)	CPU (s)
Meetjesland	14.2	4	33.1	0.26	7.3	48	48
Waasland	28.1	8	68.8	0.35	14.9	47	91
Vlaamse Ardennen	27.4	8	67.3	0.31	14.7	46	191
Meetjesland + Waasland	69.9	17	259.4	0.51	42.9	39	1367
Scheldeland	77.6	18	284.4	0.51	46.4	40	961
Scheldeland + Waasland	101.4	25	364.5	0.50	61.4	39	4717
Scheldeland + Vlaamse Ardennen	136.6	32	629.6	0.61	95.0	30	8438
Entire network	187.8	47	798.3	0.58	126.8	32	44 686
Average				0.50		40	

6. Conclusions and further research

The company Tele Atlas utilises mobile mapping vans to take pictures of streets and road signs. The mobile mapping van problem, optimising the route of the vans and taking into account the position of the sun, can be converted to an asymmetric vehicle routing problem with soft time windows. A two step metaheuristic is designed to solve this problem efficiently. Local search moves reduce the number of days and iterated local search reduces the time window violations. Hybridising the ILS metaheuristic with a linear programming solver significantly increases the quality of the results. Experimental results on a test set from the literature and some large real-life problems indicate the strength of the algorithm.

In the future, the computation time to solve large problems should be reduced by speeding up solving the linear problem. Two extra challenges should be handled in order to deal even better with real-life problems. The algorithm should be extended to tackle multiple time windows, because some roads can be photographed during the morning and during the afternoon. Furthermore, the driver of the mobile mapping van should be allowed to stay overnight in any hotel in the area, which implies that any hotel can be selected as a starting or ending point of any day. Finally we plan to extend the algorithm to a genuine multi-objective metaheuristic in which we no longer combine the two objectives in one objective function, but rather generate a set of non-dominated solutions from which Tele Atlas can choose.

Acknowledgement

Pieter Vansteenwegen is a post-doctoral research fellow of the “Fonds Wetenschappelijk Onderzoek—Vlaanderen (FWO)”.

References

[1] Aminu UF, Eglese RW. A constraint programming approach to the Chinese postman problem with time windows. Computers & Operations Research 2006;33(12):3423–31.
[2] Baldacci R, Maniezzo V. Exact methods based on node-routing formulations for undirected arc-routing problems. Networks 2006;47:52–60.
[3] Belenguer JM, Benavent E, Lacomme P, Prins C. Lower and upper bounds for the mixed capacitated arc routing problem. Computers & Operations Research 2006;33(12):3363–83.
[4] Brandão J, Eglese RW. A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research 2008;35(4):1112–26.
[5] Bräysÿ O, Gendreau M. Vehicle routing problem with time windows, part I: route construction and local search algorithms. Transportation Science 2005;39(1):104–18.
[6] Bräysÿ O, Gendreau M. Vehicle routing problem with time windows, part II: metaheuristics. Transportation Science 2005;39(1):119–39.
[7] Calvete HI, Galé C, Oliveros MJ, Sánchez-Valverde B. A goal programming approach to vehicle routing problems with soft time windows. European Journal of Operational Research 2007;177(3):1720–33.
[8] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. Cambridge, MA: MIT Press; 2001.
[9] Hashimoto H, Yagiura M, Ibaraki T. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. Discrete Optimization 2008;5(2):434–56.
[10] Ibaraki T, Imahoria S, Kubo M, Masuda T, Uno T, Yagiura M. Effective local search algorithms for routing and scheduling with general time-window constraints. Transportation Science 2005;39(2):206–32.
[11] Ioannou G, Kritikos M, Prastacos G. A problem generator-solver heuristic for vehicle routing with soft time windows. Omega 2003;31(1):41–53.
[12] Labadi N, Prins C, Reghioui M. GRASP with path relinking for the capacitated arc routing problem with time windows. Advances in Computational Intelligence in Transport, Logistics, and Supply Chain Management 2008: 111.
[13] Longo H, de Aragão MP, Uchoa E. Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research 2006;33(6):1823–37.
[14] Lourenço H, Martin O, Stützle T. Iterated Local Search 2003: 321–53.
[15] Min H. A multiobjective vehicle routing problem with soft time windows: the case of a public library distribution system. Socio-Economic Planning Science 1991;25(3):179–88.
[16] Or I. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. PhD thesis, Northwestern University, Evanston, IL; 1976.
[17] Pearn W, Assad A, Golden BL. Transforming arc routing into node routing problems. Computers & Operations Research 1987;14(4):285–8.
[18] Potvin JY, Rousseau JM. An exchange heuristic for routing problems with time windows. Journal of the Operational Research Society 1995;46:1433–46.
[19] Savelsbergh M. The vehicle routing problem with time windows: minimizing route duration. Journal of Computing 1992;4:146–54.
[20] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 1987;35(2):254–65.
[21] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin J-Y. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation science 1997;31(2):170–86.