Fast Upper and Lower Bounds for a Large-Scale Real-World Arc Routing Problem

Burak Boyacı* Thu Huong Dang† Adam N. Letchford*

December 2021, Revised April 2022

Abstract

Arc Routing Problems (ARPs) are a special kind of Vehicle Routing Problem (VRP), in which the demands are located on edges or arcs, instead of nodes. There is a huge literature on ARPs, and a variety of exact and heuristic algorithms are available. Recently, however, we encountered some real-life ARPs with over ten thousand roads, which is much larger than those usually considered in the literature. For these problems, we develop fast upper- and lower-bounding procedures. We also present extensive computational results.

Keywords: vehicle routing; arc routing; combinatorial optimisation; integer programming

1 Introduction

The optimisation of vehicle routes is of crucial importance in modern society, and there is a huge literature on models, theory, applications and algorithms [23, 43]. *Arc Routing Problems* (ARPs) are a special kind of vehicle routing problems, in which the demands are located along the edges or arcs of the network, rather than at the nodes. Typical applications of ARPs include postal delivery, meter reading, refuse collection, salt spreading and snow removal (see the books [17, 18] and the surveys [16, 35]).

Recently, while working with an industrial partner, we encountered some very challenging real-life ARPs. These problems had multiple vehicles, capacity constraints, intermediate facilities, a time deadline, multiple objectives, and a combination of one- and two-way streets. Moreover, some instances had over ten thousand roads, which is much larger than those usually considered in the literature.

*Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK. E-mail: {B.Boyaci,A.N.Letchford}@lancaster.ac.uk

†STOR-i Centre for Doctoral Training, Lancaster University, Lancaster LA1 4YR, UK. E-mail: T.H.Dang@lancaster.ac.uk

To further complicate matters, the industrial partner wanted a procedure that could produce reasonably good upper and lower bounds within a couple of minutes. This was for three reasons:

- Such a procedure could enable the sales representatives to give convincing demonstrations in real-time, to attract new customers.
- The expert trip planners in the company had stated that they would find tight initial bounds extremely useful, so that they knew what to aim for.
- It was hoped that the planners could take the feasible solutions from our procedure as a starting point, and then make adjustments to make the trips more "visually attractive" (see [41] for a discussion of visual attractiveness in vehicle routing).

The ARP in question is a (bi-objective) *Mixed Capacitated Arc Routing Problem with Intermediate Facilities and a Deadline*, or MCARPIFD for short. Although there exist a few heuristics for the MCARPIFD in the literature [30, 33, 34, 46, 47, 48], the severe restriction on computing time meant that we could not use any of them. Accordingly, we devised our own procedures, which are specially tailored to give good bounds as quickly as possible for real-life instances. We were pleased to discover that our procedures were highly suitable for the intended application.

The rest of the paper is structured as follows. Subsection 1.1 presents our notation and terminology. Section 2 is a brief literature review. Sections 3 and 4 describe our upper-bounding and lower-bounding procedures, respectively. Section 5 presents the computational results. Finally, Section 6 contains some concluding remarks.

1.1 Notation and terminology

We are given a mixed graph $G = (V, E \cup A)$, where V is the vertex set, E is the set of (undirected) edges, and A is the set of (directed) arcs. This graph represents a road network. The nodes are numbered 1 to n and have known coordinates. Node 1 is called the *depot*. We are also given a set $E_R \subseteq E$ of *required edges*, a set $A_R \subseteq A$ of *required arcs*, and a set $I \subset V$ of *intermediate facilities*. We call $L = E \cup A$ the set of *links*, and let L_R denote $E_R \cup A_R$. We will call the weakly connected components of the graph (V, L_R) "R-components".

Each link $\ell \in L$ has a positive rational *traversal time* t_ℓ. Each required link $\ell \in L_R$ has a positive rational *supply* q_ℓ and *servicing time* s_ℓ. A fleet of identical vehicles is located at the depot, each with positive rational *capacity* Q and *time limit* T. If a vehicle is used on any given day, it must depart from the depot, service some required links, go to an intermediate facility to unload, service some more required links, and so on. When the vehicle has

unloaded for the last time, it must return to the depot. Each required link must be serviced by exactly one vehicle. The load of each vehicle must not exceed Q at any time, and each vehicle must return to the depot no more than T hours after it departed.

The problem has two objective functions. The primary objective is to minimise the number of trips, but a secondary objective is to minimise the maximum trip duration. The reason for the second objective is that, if one trip has a significantly longer duration than another, then the drivers may complain that the solution is unfair. An additional peculiarity of the problem is that the number of trips must be a multiple of some given positive integer h. For example, if collections are made Monday to Friday, and each household has a collection once every two weeks, then h is set to 10.

Traversing a link without servicing is called "deadheading". The set of nodes incident on at least one required link is denoted by V_R. Given a set $S \subseteq V$, $L(S)$ denotes the set of links with both end-nodes in S, and $\delta(S)$ denotes the set of edges with exactly one end-node in S. We let $\delta^+(S)$ and $\delta^-(S)$ denote the set of arcs leaving and entering S, respectively, and $\Delta(S)$ denotes $\delta(S) \cup \delta^+(S) \cup \delta^-(S)$. We let $L_R(S)$ denote $L(S) \cap L_R$, and similarly for $\delta_R(S)$, $\delta_R^+(S)$, $\delta_R^-(S)$ and $\Delta_R(S)$. For simplicity, we sometimes write $\delta(v)$ instead of $\delta(\{v\})$. A node v is called "R-odd" if $|\Delta_R(v)|$ is odd. Finally, given a set $F \subset L$ and a vector $x \in \mathbb{R}^L$, we let $x(F)$ denote $\sum_{\ell \in F} x_\ell$.

2 Literature Review

The literature on arc routing is vast. For the sake of brevity, we cover here only papers of direct relevance. For further details, the reader is referred to the books [17, 18] and the surveys [16, 35].

2.1 The Capacitated Arc Routing Problem

Golden & Wong [24] introduced the *Capacitated Arc Routing Problem* or CARP. It is simpler than our problem, since (a) A is empty; (b) there are no intermediate facilities, and (c) there is no time deadline. Instead of times s_e and t_e, we are given a *deadheading cost* c_e for each $e \in E$. The objective is simply to minimise the total deadheading cost.

Golden and Wong showed that the CARP is $\mathcal{NP}$-hard in the strong sense. Current exact methods can cope only with instances with up to around 180 required edges; see [7] for a survey. For larger instances, various heuristics and lower-bounding procedures are available; see [15, 40, 45, 49] and [1, 4, 9, 32], respectively.

Among the many heuristics, we mention the one of Ulusoy [44]. It is a "route-first cluster-second" heuristic (see [5, 10]). The idea is to construct a single "giant" tour, that visits all of the required links, and then "split" this giant tour into segments that can be traversed by a single vehicle. The giant

tour is constructed using a heuristic, but the splitting is done optimally, via a series of shortest-path problems. An improved version of this heuristic, suitable for large-scale instances, was given in [49].

Among the many lower-bounding procedures, we will be interested in the following linear programming (LP) relaxation, which was proposed independently by Letchford [28] and Belenguer & Benavent [6]. For each $e \in E$, let y_e be a general integer variable, representing the total number of times edge e is deadheaded. Given a set $S \subseteq V \setminus \{1\}$, define

$$k(S) = \left\lceil \frac{\sum_{e \in E_R(S) \cup \delta_R(S)} q_e}{Q} \right\rceil .$$

Note that $k(S)$ is a lower bound on the number of vehicles that need to go from $V \setminus S$ to S. A valid lower bound for the CARP is then obtained by solving the following LP (either exactly or approximately) with a cutting-plane algorithm:

$$\begin{array}{lrll} \min & \sum_{e \in E} c_e y_e & & \\ \text{s.t.} & y(\delta(S)) \geq 2k(S) - |\delta_R(S)| & (S \subseteq V \setminus \{1\}) & (1) \\ & y(\delta(S)) \geq 1 & (S \subseteq V \setminus \{1\} : |\delta_R(S)| \; odd) & (2) \\ & y_e \in \mathbb{R}_+ & (e \in E). & \end{array}$$

The constraints (1) and (2) are called *capacity* inequalities and *R-odd-cut* inequalities, respectively.

We remark that the y variables are "aggregated" over all trips, and therefore do not give us information about individual trips. As a result, one cannot obtain a valid formulation of the CARP by adding an integrality constraint to the above LP.

Separation routines for the capacity and R-odd cut inequalities can be found in [6, 9, 32]. We remark that Martinelli *et al.* [32] used the inequalities within a dual ascent scheme instead of a cutting-plane algorithm. In this way, they could compute strong lower bounds for instances with a few hundred nodes and edges in reasonable computing times.

2.2 Other relevant ARPs

Li [29] considered a variant of the CARP in which all required edges must be serviced by a given time deadline. (We call this the CARPD.) He devised a simple constructive heuristic, along with a lower-bounding procedure based on the solution of a series of matching problems. Letchford [28] obtained improved lower bounds for the same problem, using cutting planes. A local search heuristic, suitable for instances with thousands of nodes and edges, was given by Wøhlk & Laporte [49]. We remark that Eglese [20] devised a heuristic for a multi-depot version of the same problem.

The mixed CARP, or MCARP, is the generalisation of the CARP in which both edges and arcs may be present. Heuristic approaches, based on constructive heuristics and genetic algorithms, were described in [8, 27]. Effective lower-bounding procedures, based on LP and cutting planes, were presented in [8, 25]. Lacomme *et al.* [27] gave a heuristic for the MCARP with turn penalties and a route-length constraint.

To our knowledge, the first paper to consider ARPs with intermediate facilities was Li & Eglese [30]. They devised a constructive heuristic for the problem that we called the MCARPIFD in the introduction. Mourão & Amado [33] presented a different constructive heuristic for the same problem, along with a lower-bounding procedure based on the solution of a transportation problem. A local search heuristic was later proposed in [34], but the heuristic was rather slow and strugged with instances having more than a few hundred nodes. Additional heuristics, suitable for instances with a few thousand nodes, were later presented by Willemse *et al.* [46, 47, 48].

Some authors have considered the undirected version of the MCARPIFD. Ghiani *et al.* [22] developed two constructive heuristics, a local search heuristic, and an LP-based lower-bounding scheme. Some additional heuristics were given in [21, 39].

We will also need a known result concerned with the *Directed Rural Postman Problem* (DRPP). The DRPP is the special case of the MCARP in which $E = \emptyset$ and there is a single vehicle with infinite capacity. If the DRPP has only a single R-component, then it can be solved in polynomial time via a reduction to an (uncapacitated) minimum-cost flow problem [19, 31].

Finally, we mention a couple of relevant papers of our own. In [12], we show that one can use Euclidean distances instead of real road distances when solving certain node routing problems, while incurring only a small loss of quality. In [13], we present a method, called *sparsification*, for improving solutions to another vehicle routing problem. We will adapt both of these ideas to our problem (see Subsections 3.1–3.4).

3 Upper Bounds

In this section, we present a fast heuristic for our version of the MCARPIFD. The heuristic is of "route-first cluster-second" type, but includes several enhancements to improve both speed and accuracy. The heuristic has seven "phases", which are described in the following subsections. Throughout this section, m denotes $|L_R|$.

3.1 Tour construction phase

Our heuristic begins with the construction of a "giant tour" through the required links. To do this, we use a procedure similar to the well-known "farthest insertion" heuristic for the TSP [42]. The differences are (a) we

have to consider the orientation of each required link when we insert it into the giant tour, and (b) we use planar Euclidean distances instead of true road network distances (as in [12]), to avoid solving all-pairs shortest path problems in G. The details are given in Algorithm 1.

Algorithm 1: Giant Tour Construction

input : Set of required links $\ell_1, \cdots, \ell_m \in L_R$, planar coordinates of all nodes in V_R

Let `GT` be an initial giant tour, in which the link ℓ_1 is traversed, and the vehicle deadheads back to its starting point;

Create a one-dimensional array of length m, called `dist`;

for $i = 2$ *to* m **do**

  Set `dist`$[i]$ to the Euclidean distance between the midpoints of ℓ_i and ℓ_1;

end

for $j = 2$ *to* m **do**

  Among all links that have not yet been inserted, let ℓ^* be a link with maximum `dist` value;

  Insert ℓ^* into `GT`, choosing the position and orientation that causes the smallest increase in the (Euclidean) length of the giant tour;

  for *each link* ℓ_i *that has not yet been inserted* **do**

    Let `Eudist`$[i]$ be the Euclidean distance between the midpoints of ℓ_i and ℓ^*;

    if `dist`$[i] >$ `Eudist`$[i]$ **then**

      Set `dist`$[i]$ to `Eudist`$[i]$;

    end

  end

end

output: Giant tour `GT`

Note that the algorithm takes as input the *planar coordinates* of each node in V_R. (We were able to get these coordinates from our industrial partner.) One can check that the algorithm runs in only $O(m^2)$ time and $O(m)$ space.

3.2 Local search phase

The giant tour is then improved, if possible, with the local search procedure described in Algorithm 2. This procedure is a variant of the well-known "λ-interchange" neighbourhood for the VRP [38], but tailored to work quickly on large-scale MCARPIFD instances. In particular, the procedure takes only $O(m^2)$ time and $O(m)$ memory.

We remark that we use Euclidean distances in Algorithm 2, instead of

Algorithm 2: Local Search

input : Set of required links $\ell_1, \cdots, \ell_m \in L_R$; planar coordinates of all nodes in V_R; giant tour `GT`

for $i = 1$ *to* m **do**
  Let ℓ'_i be the ith link in the giant tour;
end
for $i = 1$ *to* $m - 1$ **do**
  for $j = i + 1$ *to* m **do**
    if *swapping* ℓ'_i *and* ℓ'_j *reduces the tour length* **then**
      swap ℓ'_i and ℓ'_j;
    end
  end
end
output: Improved giant tour

real road network distances, just as in the previous subsection. Moreover, if ℓ'_i and/or ℓ'_j are edges, then we consider all possible orientations when evaluating the potential benefit of a swap.

3.3 Shortest-path phase

In the first two phases of our heuristic, we used Euclidean distances to estimate the amount of deadheading between consecutive links in the giant tour. The next step is to replace those Euclidean distances with the true road distances.

As before, assume that the ith link in the giant tour is ℓ'_i. The giant tour starts by servicing ℓ'_1, goes to service ℓ'_2, and so on. For $i = 1, \ldots, m - 1$, we have to compute the shortest path in G from the end-node of ℓ'_i to the start-node of ℓ'_{i+1}. (We also have to compute the shortest path from the end-node of ℓ'_m to the start-node of ℓ'_1.) To do this, for a given i, we use Dijkstra's algorithm, with a binary heap to update distance labels (see, e.g., [2]).

In theory, each shortest-path call takes $O(|L| \log |V|)$ time. In practice, however, it is extremely fast, since (a) the end-node of ℓ'_i and the start node of ℓ'_{i+1} are frequently identical, and (b) we can abort a given call as soon as the start-node of ℓ'_{i+1} becomes permanently labelled.

In the fifth phase of our heuristic (Subsection 3.5), we will need to know the distance from node 1 (the depot) to each node in V_R, from each node in V_R to each node in I, and from each node in I to each node in $V_R \cup \{1\}$. To calculate all these values, we run Dijkstra's algorithm $2|I| + 1$ more times. This takes $O(|L|\,|I| \log |V|)$ time in total. Fortunately, $|I|$ was less than five in the instances that our client encountered.

3.4 Sparsification phase

In the fourth phase of our heuristic, we attempt to reduce the length of the giant tour. The procedure is essentially an extension of the "sparsification" method in [13] to the case of mixed graphs. The procedure is rather complicated, but we have found that it is very worthwhile, typically leading to reductions in tour length of over 10% in just a few seconds.

The next step is to construct a digraph $G' = (V, A^1 \cup A^2)$. This is done as follows. Initially $A^1 = A_R$ and $A^2 = \emptyset$. Then, for each required edge $\{i, j\}$ in E_R, we add the arc (i, j) or (j, i) to A^1, according to the direction in which it is traversed in the giant tour. After that, we do the following for each deadheading arc in the giant tour: we take the corresponding shortest dipath in G, and add the arcs in that dipath to A^2. Finally, we remove from A^2 any arc whose end-nodes lie in the same R-component. Note that G' is weakly connected.

We now construct a smaller undirected graph, which we call the "shrunk graph" and denote by G_S. To do this, we take G', shrink each R-component into a single (required) node, delete all loops, and ignore the directions of the arcs in A^2. We also delete all nodes of degree zero. Note that G_S is connected.

Next, we compute a *Minimum Spanning Tree* (MST) in G_S, and delete all edges that are not in the tree from G_S. We then check if there are any non-required nodes in G_S that have degree one. Any such node is deleted from G_S, along with the incident edge, and this is done iteratively until no such nodes remain. Note that G_S remains connected.

We now return to the digraph G', and construct a "sparsified" version of it. Specifically, given any edge that was removed from G_S, we remove the corresponding arc from A^2. Note that, by construction, G' now contains one huge weakly connected component that contains all nodes in V_R and all arcs in A^1. On the other hand, this component typically does not represent a tour, due to the presence of "unbalanced" nodes (i.e., nodes for which the number of incoming arcs is not equal to the number of outgoing arcs).

To find the minimum-cost amount of extra deadheading needed to make all nodes balanced, we solve a Directed Rural Postman Problem (DRPP). Due to the result mentioned in Subsection 2.2, this DRPP can be reduced to an uncapacitated minimum-cost flow problem in G. This problem in turn can be reduced to $O(|V| \log |V|)$ shortest-path problems in G (see [37]). For ease of implementation, we used the `MinCostFlow` solver from Google's "OR-Tools"[1]. For the instances we tested, it was extremely fast.

Adding the additional deadheading arcs to G', we obtain a strongly connected and balanced component that contains all arcs in A'. Thus, the component represents a giant tour. To construct the giant tour explicitly, we use Hierholzer's algorithm [26], which takes only $O(|L|)$ time.

[1] `http://developers.google.com/optimization`

The sparsification phase is illustrated in Figure 1. First consider the mixed graph G in Figure 1(a). Required and non-required links are represented by thick and thin lines, respectively. (In the online version of the paper, the thick lines are red.) The costs are also indicated on the links. Suppose that the first two phases have produced a giant tour which traverses the required links in the order $(1,2),(3,4),(8,9),(9,10),(13,12),(12,11)$. Figure 1(b) shows the corresponding digraph G', *before* we remove any arcs from A^2. We see that the arc $(12,11)$ is in A^2, yet both of its nodes are in the same R-component. So, this arc will be removed from G'. Figure 1(c) shows G' after this arc removal, and Figure 1(d) shows the shrunk graph G_S. The edges in the MST on G_S are indicated by red thick lines in Figure 1(e). Now, node 6 is a non-required node with unit degree, so it will be removed from the MST. Figure 1(f) represents the reduced MST on G_S. Figure 1(g) represents the sparsified version of G', which only contains required arcs and deadheading arcs corresponding to edges in the reduced MST. One can check that links $(4,7)$, $(7,12)$ and $(11,8)$ form the minimum cost amount of extra deadheading needed to make all nodes balanced. Figure 1(h) shows the new giant tour in G. It can be checked that the new giant tour costs 3 less than the old one.

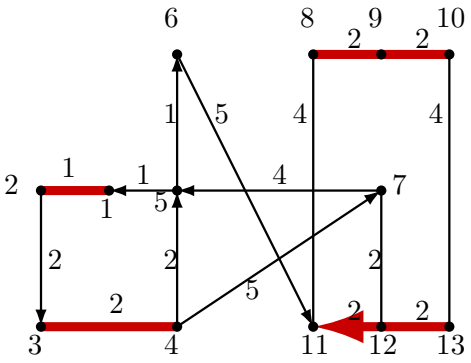
3.5 Trip construction phase

Recall that we have a vehicle capacity Q and time limit T. In the next phase of the algorithm, we construct a collection of "potential" trips that satisfy both of these restrictions. In this phase, the indices of the links in the giant tour are taken modulo m. In other words, the link ℓ'_i may also be called ℓ'_{i+m}.

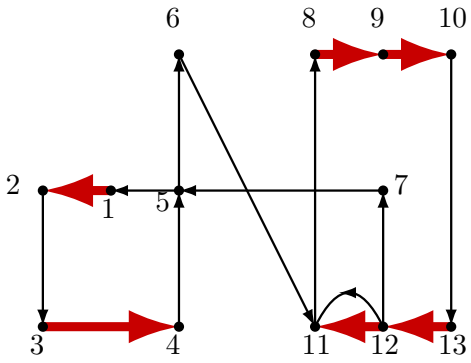
For $i = 1,\ldots,m$, a trip is created as follows. The vehicle departs from the depot, deadheads to the start node of ℓ'_i, and then services ℓ'_i, ℓ'_{i+1} and so on, until it cannot service any more links due to the capacity limit. At this point, it visits a node in I to unload. The vehicle then continues to service links and periodically visit nodes in I, until it cannot service any more links without violating the time limit. At that point, it unloads one last time at a node in I and returns to the depot.

Our method to generate trips is similar to the "first-fit bin-packing heuristic" in [47]. The heuristic takes only $O(m)$ time for a given i, which makes the running time of this phase $O(m^2)$ in total. Once all m trips are created, we let len$[i]$ denote the number of required links that are serviced in the ith trip.

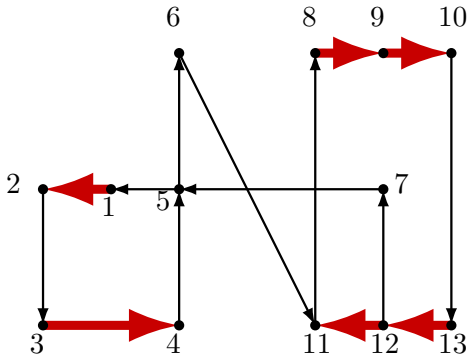
For reasons which will become clear in Subsection 3.7, we compute and store some additional information. Specifically, for a given i, and for $\beta \in \{5,10,15\}$, we let len$[i,\beta]$ be the number of links that would be serviced in the ith trip, if the time limit were decreased by β minutes. Note that the additional computing time and memory that is required to calculate and



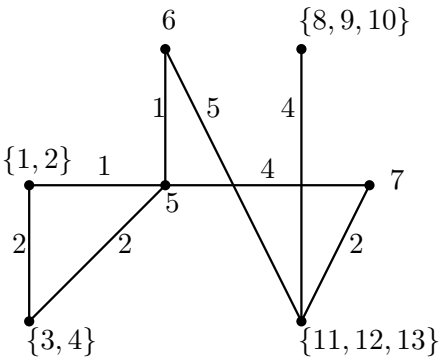
(a) Original mixed graph G



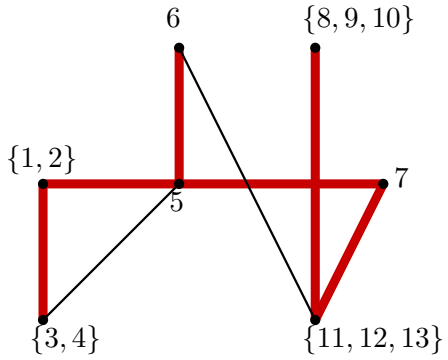
(b) digraph G' before arc removal



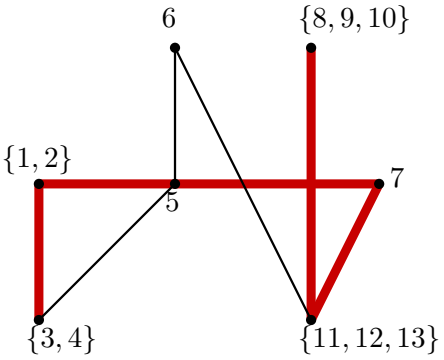
(c) Digraph G' after arc removal



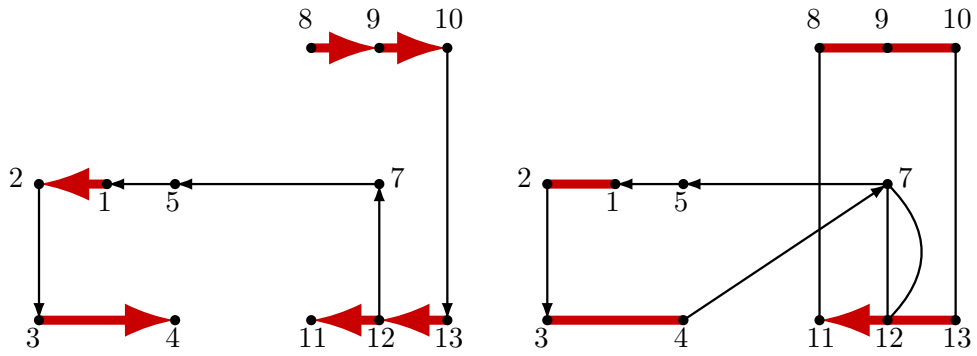
(d) Graph G_S



(e) MST on graph G_S



(f) MST with redundant edge removed



(g) Sparsified version of digraph G' (h) New giant tour

Figure 1: An illustration for the sparsification phase

store this extra information is negligible.

3.6 Trip selection phase

Next, we attempt to construct an MCARPIFD solution that uses as few trips as possible, but we temporarily ignore the fact that the number of trips must be a multiple of h. For this, we use Algorithm 3.

The algorithm constructs m MCARPIFD solutions, and stores the best along the way. The asterisk indicates the best solution found so far (sometimes called the "incumbent"). The number of trips used in the incumbent is denoted by N.

In the ith solution, the first link to be serviced by the first vehicle is ℓ'_i and the last link to be serviced by the last vehicle is ℓ'_{i+m-1} (with indices again being taken module m). The index k represents the number of trips that have been selected so far (for the given i), and cu[k] represents the cumulative number of required links that have been serviced by those k trips.

Algorithm 3 takes only $O(m^2)$ time and $O(m)$ space. It is similar to the fastest tour-splitting procedure in [47], the difference being the addition of the "backtracking" step. This loop allows us to see if a saving can be made by "backtracking" along the kth trip. That is, while considering the kth trip, we check if there exists an index j, with $cu[k-1] < j \leq cu[k]-1$, such that ending the kth trip at ℓ'_{i+j} and starting the $(k+1)$th trip from ℓ'_{i+j+1} increases the cumulative number of required links serviced so far. We found that this extra step frequently leads to a reduction in the number of trips, with negligible additional computing effort.

To see how backtracking works, suppose len[1] = 10, len[10] = 11 and len[11] = 9. When $i = 1$ and $k = 1$, before we start backtracking, we obtain

cu[1] = len[1] = 10 and cu[2] = cu[1] + len[11] = 10 + 9 = 19. So, the first trip in our solution services 10 links and the second services 9. Now consider what happens when we apply backtracking. When $j = 9$, we have $j+$ len$[i + j] = 9 + 11 = 20 > 19 =$ cu[2]. Thus, we reduce cu[1] from 10 to 9 and increase cu[2] from 9 to 11. After this change, the first trip services 9 links and the second services 11. Thus, the total number of links serviced by the first two trips has increased from 19 to 20.

3.7 Time reduction phase

The procedure in the previous subsection yields a collection of N trips. Let $r = N \bmod h$. If $r = 0$, we have a feasible MCARPIFD solution, and our upper-bounding procedure ends. If $r \neq 0$, however, then we do some more work. Let $U = h\lceil N/h \rceil$. By definition, U is an upper bound on the minimum number of trips needed. Moreover, there is a chance that, if we use U trips instead of N, we will be able to reduce the maximum trip duration.

To deal with this, we use the "len[i,β]" values, that we mentioned at the end of Subsection 3.5. In more detail, for $\beta \in \{5, 10, 15\}$, we re-run Algorithm 3, with the "len[i]" values replaced with the "len[i,β]" values. If we find a solution that uses no more than U trips, but has a smaller maximum trip duration, we replace the old solution with the new solution.

For example, suppose that $T = 8$, $h = 10$ and $N = 33$. We have $U = 40$ and $r = 3$. Suppose that the number of trips used for $\beta = 5, 10$ and 15 is 35, 38 and 41, respectively. We now have an MCARPIFD solution that uses no more than 40 trips and has a maximum trip duration of no more than 7 hours and 50 minutes. (Note that the maximum trip duration in that particular solution may well be even smaller than that.)

4 Lower Bounds

In this section, we present a lower-bounding algorithm for the MCARPIFD. This algorithm is specifically designed to give bounds of acceptable quality in just a couple of minutes.

The algorithm actually computes lower bounds on three quantities: the total number of trips, the total number of visits to intermediate facilities, and the total travel time. We denote these bounds by K, α and τ, respectively. The following six subsections present the key components of our algorithm.

4.1 Initial lower bounds

The first step is to compute initial estimates of K, α and τ. To do this, we use Algorithm 4. In this algorithm, $t(\ell)$ denotes the minimum time needed to travel from the depot to an end-node of link ℓ, $t'(\ell)$ denotes the minimum time needed to travel from an end-node of ℓ to the nearest node in I, $t''(\ell)$

denotes the minimum time needed to travel from a node in I to an end-node of ℓ, and $t'''(\ell)$ denotes the minimum time needed to travel from an end-node of ℓ to the depot via a node in I.

Lemma 1. *The bounds produced by Algorithm 4 are valid.*

Proof. Let K' be the minimum number of trips in an optimal solution. If $K' = K$, then the total number of times a vehicle travels from a required link to a node in I is at least $\alpha - K$, and the same is true for the number of times a vehicle travels from a node in I to a required link. The result is then immediate.

Suppose instead that $K' > K$. Consider one specific trip. It forms a closed walk in G that includes the depot and passes through an intermediate facility at least once. Suppose we short-cut this closed walk, by omitting the depot, and then add the resulting closed walk to one of the other trips. The cost of the resulting (possibly infeasible) solution is no larger than that of the original. Repeating this argument, if necessary, we obtain a (possibly infeasible) solution that costs no more than the optimal solution, but uses only K trips. We can then apply the argument in the preceding paragraph. □

4.2 Auxiliary digraph

For what follows, it is helpful to define an auxiliary directed graph, which we denote by $\bar{G} = (\bar{V}, \bar{A})$. This digraph is created as follows. Initially, $\bar{G}$ is a copy of G. We then replace each edge $e \in E$ with a pair of directed arcs, one in each direction.

Next, we add some "dummy" nodes and arcs. We add a node 1^* and a node set I^*, which can be thought of as copies of the depot and intermediate facilities, respectively. For each node $v \in V_R$, we add the arc $(1^*, v)$ to $\bar{A}$. (This arc represents the journey from the depot to the v.) For each node $v \in V_R$ and each dummy node $i^* \in I^*$, we add the arcs (v, i^*) and (i^*, v) to $\bar{A}$. (These arcs represent journeys from v to an intermediate facility, or vice-versa.) Finally, for each $i^* \in I^*$, we add the arc $(i^*, 1^*)$ to $\bar{A}$. (These arcs represent journeys from intermediate facilities to the depot, at the end of the day.)

For each dummy arc $(u, v) \in \bar{A}$, we let t_{uv} denote the time taken to travel from u to v in G if one follows the quickest path. To compute these times, it suffices to call Dijkstra's single-source shortest-path algorithm $|I| + 1$ times in G. This takes $O(|I|(|L| + |V| \log |V|))$ time.

For notational ease, we identify t_{uv} and t_{vu} for each edge $\{u, v\} \in E$, and we identify s_{uv} and s_{vu} for each edge $\{u, v\} \in E_R$. We also let $\bar{\delta}^+(S)$ and $\bar{\delta}^-(S)$ denote the set of arcs in $\bar{G}$ leaving and entering S, respectively. Finally, we let $\bar{A}_R$ denote the set of arcs in $\bar{A}$ that represent arcs in L_R.

(That is, there are two arcs in $\bar{A}_R$ for each edge in E_R, plus one arc for each arc in A_R.)

4.3 Initial LP relaxation

Our initial LP relaxation is an extension of the one for the CARP mentioned in Subsection 2.1. It uses two kinds of variables, called x and y.

For each edge $\{u, v\} \in E_R$, the binary variables x_{uv} and x_{vu} indicate the direction in which $\{u, v\}$ is serviced. (For notational simplicity, we also define a variable x_{uv} for each arc $(u, v) \in A_R$. These latter variables are fixed to 1.) For each arc $(u, v) \in \bar{A}$, the general-integer variable y_{uv} represents the total number of times that (u, v) is deadheaded.

We remark that the y variables for the dummy arcs take into account any deadheading that occurs while vehicles are (a) on the way from the depot to the first link that they service, (b) travelling to and from the intermediate facilities, or (c) returning to the depot at the end of the day. The y variables for the remaining arcs deal with any additional deadheading.

The initial LP is then as follows:

$$\sum_{a \in \bar{A}} t_a y_a \tag{3}$$

$$x_{uv} + x_{vu} = 1 \qquad (\{u, v\} \in E_R) \tag{4}$$

$$x_{uv} = 1 \qquad ((u, v) \in A_R) \tag{5}$$

$$y(\bar{\delta}^+(1^*)) \geq K \tag{6}$$

$$y(\bar{\delta}^+(I^*)) \geq \alpha \tag{7}$$

$$x(\bar{\delta}_R^+(v)) + y(\bar{\delta}^+(v)) = x(\bar{\delta}_R^-(v)) + y(\bar{\delta}^-(v)) \qquad (v \in V_R) \tag{8}$$

$$y(\bar{\delta}^+(v)) = y(\bar{\delta}^-(v)) \qquad (v \in \bar{V} \setminus V_R) \tag{9}$$

$$y_{1^*,v} + \sum_{i^* \in I^*} (y_{v,i^*} + y_{i^*,v}) \leq |\delta_R(v)| \qquad (v \in V_R) \tag{10}$$

$$x_{uv}, x_{vu} \geq 0 \qquad (\{u, v\} \in E_R)$$

$$y_{uv} \geq 0 \qquad ((u, v) \in \bar{A}).$$

The objective function (3) represents the total amount of time spent deadheading. Constraints (4) and (5) ensure that each required link is serviced. Constraint (6) ensures that at least K trips are used. Constraint (7) ensures that the intermediate facilities are visited at least α times. Constraints (8) and (9) ensure that the number of vehicles leaving each node is equal to the number of vehicles arriving. Constraints (10) arise due to the fact that each node in V_R is incident on a small number of required links. The remaining constraints are trivial.

We remark that, due to the sparsity of road networks, our initial LP contains only $O(n)$ constraints. In practice, it can be solved in a few seconds, even for very large instances.

4.4 Cutting-plane algorithm

Recall that the industrial partner wanted lower bounds to be available within a couple of minutes. Since our initial LP was typically solved in a few seconds, we had some spare time. This led us to devise a cutting-plane algorithm, which uses analogues of the capacity inequalities (1) and R-odd-cut inequalities (2).

To this end, we define:

$$k(S) = \left\lceil \frac{\sum_{\ell \in L_R(S) \cup \Delta_R(S)} q_\ell}{Q} \right\rceil \qquad (\emptyset \neq S \subseteq V).$$

The analogue of the capacity inequalities is then:

$$y(\bar{\delta}^+(S) \cup \bar{\delta}^-(S)) \geq 2k(S) - |\Delta_R(S)| \qquad (\emptyset \neq S \subseteq V). \tag{11}$$

The analogue of the R-odd-cut inequalities is:

$$y(\bar{\delta}^+(S) \cup \bar{\delta}^-(S)) \geq 1 \qquad (S \subset V : |\Delta_R(S)| \text{ odd}). \tag{12}$$

For a high-level description of our algorithm, refer to Algorithm 5. Note that, each time the LP is re-optimised, we check whether any of K, α or τ can be increased.

Due to the limited computation time available, we do not use sophisticated separation algorithms. Instead, we use the following fast and simple heuristic:

1. Let y^* be the value of the vector y at the current LP solution.
2. Let $\epsilon = 0.01$.
3. Construct a graph $G^* = (V, E^*)$ as follows. For each arc $(u, v) \in \bar{A}$ such that $\{u, v\} \subset V$ and $y^*_{uv} \geq \epsilon$, the edge $\{u, v\}$ is included in E^*.
4. Check whether G^* is connected. If not, check each connected component, to see if it violates a capacity inequality (11) or R-odd cut inequality (12).
5. Expand E^* as follows. For each arc $(u, v) \in \bar{A}_R$ such that the edge $\{u, v\}$ is not already in E^*, insert $\{u, v\}$ into E^*.
6. Repeat step 4.

Our implementation of this heuristic runs in $O(|V|\,|L|)$ time (because there are $O(|V|)$ components, and checking each one takes $O(|L|)$ time).

In our preliminary computational tests, we found that the cutting-plane algorithm exhibited a pronounced "tailing-off" effect. That is, the lower bound improved rapidly in the early iterations, but extremely slowly after that. For this reason, we run the algorithm for one minute only for any given instance.

4.5 Additonal flow variables

Observe that the cutting planes (11) take the vehicle capacity into account, but not the time deadline. In principle, it is possible to strengthen the right-hand side of (11) by taking time into account. However, we could not find a fast algorithm for doing this. Instead, we found it more effective to introduce additional variables and constraints to represent the flow of time.

Assume without loss of generality that all trips begin at time 0. For each arc $(u, v) \in \bar{A}$, let f_{uv} be a non-negative continuous variable, with the following interpretation. If the arc is not traversed by any vehicle, then f_{uv} takes the value 0. If the arc is traversed exactly once, then f_{uv} represents the time at which the corresponding vehicle departs from node u. If the arc is traversed several times, then f_{uv} represents the sum of the corresponding elapsed times. In other words, the f variables are "aggregated" over all trips, just like the y variables.

Before presenting the constraints, we need a little more notation. For each node $u \in \bar{V}$, let $e(u)$ and $\ell(u)$ be the earliest and latest times at which a vehicle can arrive at node u, or depart from node u, respectively. (These values can be computed with two calls to Dijkstra's algorithm.)

We then add the following constraints to the LP:

$$
\begin{aligned}
f(\bar{\delta}^+(v)) = f(\bar{\delta}^-(v)) + \textstyle\sum_{a \in \bar{\delta}^-(v)} t_a y_a \qquad & (v \in \bar{V} \setminus (V_R \cup \{1^*\}) \\
f(\bar{\delta}^+(v)) = f(\bar{\delta}^-(v)) + \textstyle\sum_{a \in \bar{\delta}_R^-(v)} s_a x_a + \sum_{a \in \bar{\delta}^-(v)} t_a y_a \qquad & (v \in V_R \setminus \{1^*\}) \\
f_{uv} \geq e(u) y_{uv} \qquad & ((u, v) \in \bar{A} \setminus \bar{A}_R) \\
f_{uv} \geq e(u)(x_{uv} + y_{uv}) \qquad & ((u, v) \in \bar{A}_R) \\
f_{uv} \leq (\ell(v) - t_{uv}) y_{uv} \qquad & ((u, v) \in \bar{A} \setminus \bar{A}_R) \\
f_{uv} \leq (\ell(v) - s_{uv} - t_{uv}) x_{uv} + (\ell(v) - t_{uv}) y_{uv} \qquad & ((u, v) \in \bar{A}_R).
\end{aligned}
$$

The interpretation of these constraints is straightforward and omitted for brevity.

5 Computational Results

For all of our computational experiments, we used a laptop with an 11th Gen Intel Core i7 processor, running under Windows 10 at 3GHz with 16GB of RAM. All algorithms were implemented with `C#` in the `.NET` framework, and compiled with Microsoft Visual Studio 2019. To solve the LP relaxations, the code called on the dual simplex solver of `CPLEX` (v. 12.10).

5.1 Test Instances

Since we were asked by the industrial partner not to share details of their MCARPIFD instances, we created some artificial instances for this paper.

We did however take care to ensure that they are realistic. In particular, we used real road network data, extracted using the Python package `OSMnx` [11].

We selected five cities: Seoul, New York, Istanbul, Hanoi and London. For each city, we selected a central landmark. After that, we did the following for each city and each value $\beta \in \{2500, 5000, 7500, 10000\}$. We computed the smallest square, centred on the landmark, that contains at least β nodes. Table 1 shows, for each of the five cities, the name of the landmark and the length (and therefore also width) of the four squares, in metres.

For each of the 20 resulting squares, we created a mixed graph G as follows. The set of links L was determined by the set of roads that were wholly contained in the square. After that, we set V to the set of nodes that were incident on at least one link in L.

Unfortunately, we found that a few of the resulting mixed graphs were not strongly connected. To deal with this, we used Kosaraju's Algorithm [3] to compute the strongly connected components for each graph. We then redefined G to be the largest component in each case. Note that, by construction, $|V|$ is always smaller than β.

The next step was to decide which links were required. For simplicity, we just made each link required with probability 1/2. Some summary statistics for the resulting 20 graphs can be found in Table 2. Note that all cities have a significant number of one-way streets.

We now describe the default parameters used. (We explore the effect of varying these parameters in Subsection 5.3). We set the time limit T to 8 hours, the vehicle capacity Q to 10.5 tonnes, and the travelling speed to 30 kph. We set h to 10, which means that each household has a weekday collection every two weeks. The depot was placed at the top-left, while two intermediate facilities were located at the top-right and bottom-left. Two-way streets were treated as edges.

The servicing times s_ℓ, measured in hours, followed a log-normal distribution. The mean and standard deviation in the log scale were set to -3.933 and 1.005, respectively. To set the demands q_ℓ, we used the regression equation $q_\ell = 0.0035 + 2.7879 s_\ell + \epsilon_\ell$, where the noise terms ϵ_ℓ were i.i.d. normal variables with mean 0 and standard deviation 0.0386. (All of these parameters were based on our experience with real instances.)

Full details of all instances will be made available at the Lancaster University Data Repository[2].

5.2 Results for the default scenario

First we present the results obtained with our upper-bounding procedure. Table 3 shows the following for each of the 20 instances: the city name, the

[2] `http://www.research.lancs.ac.uk/portal/en/datasets/search.html`

number of nodes ($|V|$), the number of trips in the heuristic solution (K), the number of visits to the intermediate facilities (α), the mean and maximum trip durations (in hours), and the computing time (in seconds).

As one might expect, the number of trips scales roughly linearly with the number of nodes, and so does the number of visits to intermediate facilities. Note that, in all instances, $\alpha \approx 2K$. This means that the vehicles tend to visit an intermediate facility twice in each trip. This is similar to what we encountered in practice.

The mean and maximum trip durations are well within 8 hours in every case. It is clear that the procedure mentioned in Subsection 3.7 has succeeded in reducing the maximum duration in all cases. Closer inspection of the output showed that, for some instances, the last trip generated had a significantly shorter duration than all of the others. This suggests that one could reduce the maximum duration further, while keeping the number of trips the same, by applying some kind of local search procedure.

As for the running times, the heuristic runs in less than 25 seconds in all cases. Our industrial partner was very happy with the upper bounds and running times.

We now turn our attention to the lower-bounding procedure. Table 4 shows the following for each instance: the city name and the number of nodes (as before); the lower bounds on the number of trips (K), number of visits to the intermediate facilities (α) and total travel time (τ); the ratio between the lower and upper bounds on K and τ; and the computing time in seconds.

We see that the lower-bounding procedure produces excellent lower bounds on K and α, and good lower bounds on τ, within one and a half minutes. Moreover, despite the severe restrictions on computing time, our procedures have found the proven optimal value of K for 18 instances. We suspect that, for the remaining two instances, one could find a solution that uses 10 fewer trips (and therefore one fewer vehicle), given additional computing time.

5.3 Sensitivity analysis

For completeness, and to aid insight, we conduct a sensitivity analysis. We experiment with varying three parameters: the number of weeks between consecutive visits to customers, the number of intermediate facilities, and the deadheading speed. We also examine the effect of having to service each required edge twice, once in each direction.

5.3.1 Visit frequency

In our default scenario, h is 10, which means that each customer is visited every two weeks. We explore the effect of changing h to 5 (corresponding to weekly visits) and 15 (corresponding to one visit every three weeks). Note

that, when h is reduced to 5, the demand for each required link is halved. Similarly, when h is increased to 15, the demands are multiplied by 1.5.

Figure 2 shows the gap between the lower and upper bounds on the number of trips K, for all 20 instances and for the three values of h. In each box, the first 5 lines represent the default setting ($h = 10$), the next 5 lines represent the case $h = 5$, and the last 5 lines represent the case $h = 15$. (In the online version, the lines are colored blue, red and green, respectively.) A line of zero length indicate that the upper and lower bounds coincide.

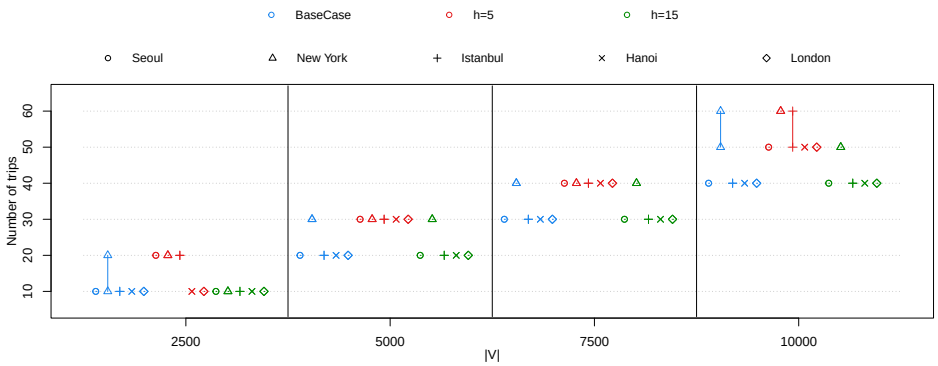


Figure 2: Estimates of K when h varies.

Remarkably, changing the value of h does not make a big difference to the number of trips. Closer inspection of the output revealed that an increase in h usually led to a significant increase in α (the number of visits to the intermediate facilities), but not much difference to K or τ. This suggests that larger values of h are much more economical for the service provider, since they significantly reduce the number of trips per week (and therefore also the number of vehicles required). We remark that some local councils in the UK recently suggested increasing h from 10 to 15, but they were forced to abandon the idea due to opposition from the public.

5.3.2 Number of intermediate facilities

In our default scenario, there are two intermediate facilities (IFs). We now explore the effect of having just one (located in the top-right of the square) or three (located at top-right, bottom-left and bottom-right).

Figure 3 is similar to Figure 2, except that the three cases are $|I| = 2, 1, 3$. It is apparent that changing the number of IFs has no effect on the number of trips in almost all cases. Closer inspection of the output showed that increasing the number of IFs led to a small increase in α, a small decrease in τ, and a small decrease in the maximum trip duration. This is because vehicles can travel to the nearest IF to unload, instead of being forced to go to one specific IF.

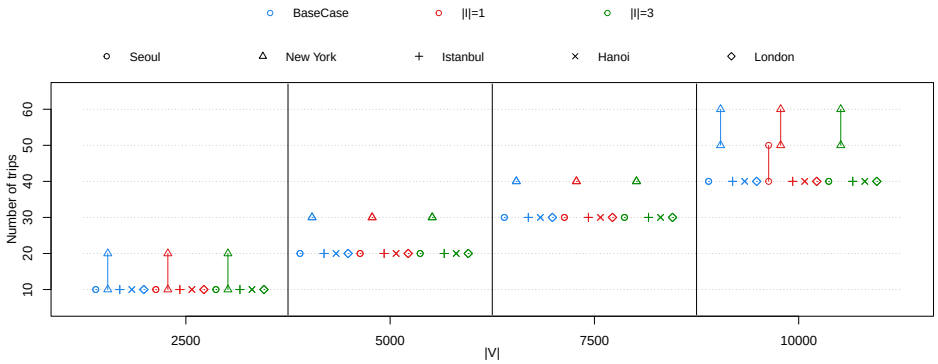


Figure 3: Estimates of K when $|I|$ varies.

5.3.3 Deadheading speed

Next, we explored the effect of changing the deadheading speed from 30kph to either 20kph or 40kph. Figure 4 shows the resulting estimates of K. As one would expect, the number of trips needed tends to decrease as the deadheading speed increases. The effect is however fairly small. This is probably because, in an urban setting, the majority of time is spent servicing rather than deadheading. For the same reason, increasing the deadheading speed tended to lead to only small decreases in τ. As for α, there was no discernable pattern.

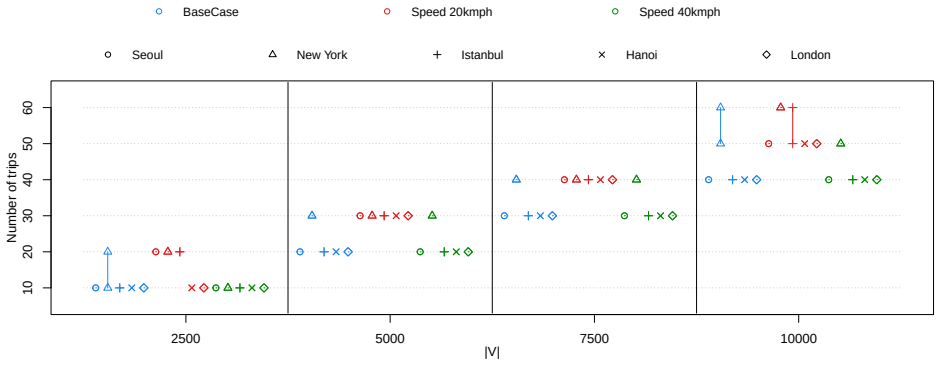


Figure 4: Estimates of K when deadheading speed varies.

5.3.4 Two-lane service

Finally, we consider the case in which each required edge must be serviced twice, once in each direction. (More precisely, each edge in E_R is replaced by a pair of anti-parallel required arcs, each with half the demand.)

Figure 5 shows the resulting bounds on K. As one might expect, requiring edges to be serviced twice tends to lead to an increase in the number of trips. Again, however, the effect is less marked than one might expect. A possible explanation is that replacing each required edge with a pair of required arcs causes the total number of R-odd nodes to decrease. As a result, the amount of deadheading tends to decrease, even if the amount of servicing increases. A similar pattern was seen with τ.

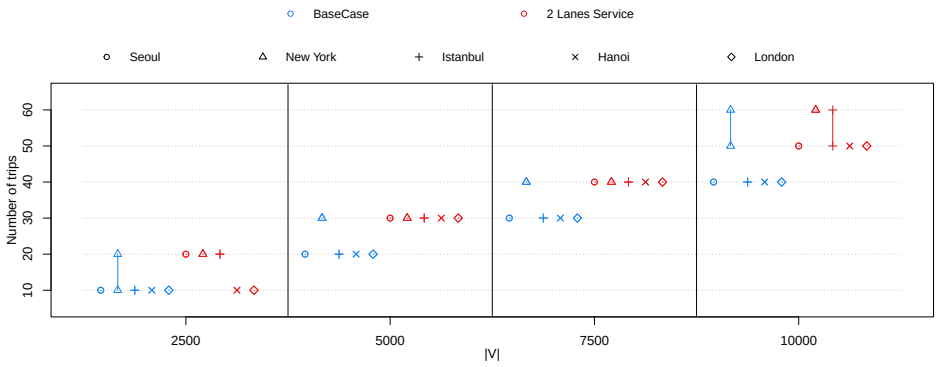


Figure 5: Estimates of K with one or two visits to required edges.

We remark that, throughout our experiments, the lower bound on τ was never less than 83% of the upper bound. In fact, in 85% of the cases, it was above 90%. We also remark that requiring edges to be serviced twice caused our upper-bounding procedure to take around twice as long. This is because the giant tour becomes nearly twice as large. On the other hand, there was no significant increase in the time taken by our lower-bounding procedure, due to the fact that we imposed a limit of one minute for the cutting-plane phase.

6 Conclusion

In this paper, we have considered a "rich" arc routing problem, with vehicle capacities, a time deadline, intermediate facilities, a mixture of one- and two-way streets, different routes on different days, and fairness considerations. For this problem, our industrial partner wanted algorithms that could produce lower and upper bounds within just a couple of minutes. We were able to accomplish this by using a judicious combination of known and new techniques.

We can think of three possible topics for future research. The first is the development of local search heuristics to improve the upper bounds obtained with our approach. The second is the development of a heuristic to decompose the problem into a number of smaller problems. (This is called

"districting" in the arc routing literature [14, 36].) The third is the development of an exact algorithm for the single-vehicle version of the problem. Such an algorithm could perhaps be used to "polish" the individual trips that are generated by our heuristic, or indeed the trips found by the expert trip planners.

Acknowledgement: The second author gratefully acknowledges financial support from Routeware, Inc., and from the EPSRC through the STOR-i Centre for Doctoral Training under grant EP/L015692/1.

References

[1] D. Ahr & G. Reinelt (2015) The capacitated arc routing problem: combinatorial lower bounds. In Corberán & Laporte (eds.), pp. 159–181.

[2] R.K. Ahuja, T.L. Magnanti & J.B. Orlin (1993) *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall.

[3] A.V. Aho, J.E. Hopcroft & J.D. Ullman (1983) *Data Structures and Algorithms*. Amsterdam: Addison-Wesley.

[4] E. Bartolini, J.F. Cordeau & G. Laporte (2013) Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Math. Program.*, 137, 409–452.

[5] J.E. Beasley (1983) Route first—cluster second methods for vehicle routing. *Omega*, 11, 403–408.

[6] J.M. Belenguer & E. Benavent (2003) A cutting-plane algorithm for the capacitated arc routing problem. *Comput. Oper. Res.*, 30, 705–728.

[7] J.M. Belenguer, E. Benavent & S. Irnich (2015) The capacitated arc routing problem: exact algorithms. In Corberán & Laporte (eds), pp. 183–221.

[8] J.-M. Belenguer, E. Benavent, P. Lacomme & C Prins (2006) Lower and upper bounds for the mixed capacitated arc routing problem. *Comput. Oper. Res.* 33, 3363–3383.

[9] E. Benavent, Á. Corberán & J.M. Sanchis (2000) Linear programming based methods for solving arc routing problems. In Dror (ed.), pp. 231–275.

[10] L.D. Bodin (1975) A taxonomic structure for vehicle routing and scheduling problems. *Comput. Urban Soc.*, 1, 11–29.

[11] G. Boeing (2017) OSMnx: new methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Comput. Environ. Urban Syst.*, 65, 126–139.

[12] B. Boyacı, T.H. Dang & A.N. Letchford (2021) Vehicle routing on road networks: how good is Euclidean approximation? *Comput. Oper. Res.*, 129, article 105197.

[13] B. Boyacı, T.H. Dang & A.N. Letchford (2021) Improving a constructive heuristic for the general routing problem. *Working paper*, Department of Management Science, Lancaster University, UK.

[14] A. Butsch, J. Kalcsics & G. Laporte (2014) Districting for arc routing. *INFORMS J. Comput.*, 26, 809–824.

[15] Y. Chen, J.K. Hao & F. Glover (2016) A hybrid metaheuristic approach for the capacitated arc routing problem. *Eur. J. Oper. Res.*, 253, 25–39.

[16] Á. Corberán, R. Eglese, G. Hasle, I. Plana & J.M. Sanchis (2021) Arc routing problems: a review of the past, present, and future. *Networks*, 77, 88–115.

[17] Á. Corberán & G. Laporte (eds) (2015) *Arc Routing: Problems, Methods, and Applications*. Philadelphia, PA: SIAM.

[18] M. Dror (ed.) (2000) *Arc Routing: Theory, Solutions and Applications*. Dordrecht: Kluwer.

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

[20] R.W. Eglese (1994) Routeing winter gritting vehicles. *Discr. Appl. Math.*, 48, 231–244.

[21] G. Ghiani, F. Guerriero, G. Laporte & R. Musmanno (2004) Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions. *J. Math. Model. Alg.*, 3, 209–223.

[22] G. Ghiani, G. Improta & G. Laporte (2001) The capacitated arc routing problem with intermediate facilities. *Networks*, 37, 134–143.

[23] B. Golden, S. Raghavan & E. Wasil (eds.) (2008) *The Vehicle Routing Problem: Latest Advances and New Challenges*. Boston, MA: Springer.

[24] B.L. Golden & R.T. Wong (1981) Capacitated arc routing problems. *Networks*, 11, 305–15.

[25] L. Gouveia, M.C. Mourão & L.S. Pinto (2010) Lower bounds for the mixed capacitated arc routing problem. *Comput. Oper. Res.*, 37, 692–699.

[26] C. Hierholzer (1873) Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6, 30–32.

[27] P. Lacomme, C. Prins & W. Ramdane-Cherif (2004) Competitive memetic algorithms for arc routing problems. *Ann. Oper. Res.*, 131, 159–185.

[28] A.N. Letchford (1997) *Polyhedral Results for Some Constrained Arc-Routing Problems*. PhD thesis, Department of Management Science, Lancaster University, UK.

[29] L.Y.O. Li (1992) *Vehicle Routeing for Winter Gritting*. PhD thesis, Department of Management Science, Lancaster University, UK.

[30] L.Y.O. Li & R.W. Eglese (1996) An interactive algorithm for vehicle routeing for winter–gritting. *J. Oper. Res. Soc.*, 47, 217–228.

[31] T. Liebling (1970) *Graphentheorie in Planungs- und Tourenproblemen*. Heidelberg: Springer.

[32] R. Martinelli, M. Poggi de Aragão & A. Subramanian (2013) Improved bounds for large scale capacitated arc routing problem. *Comput. Oper. Res.*, 40, 2145–2160.

[33] M.C. Mourão & M.T. Almeida (2000) Lower-bounding and heuristic methods for a refuse collection vehicle routing problem. *Eur. J. Oper. Res.*, 121, 420–434.

[34] M.C. Mourão & L. Amado (2005) Heuristic method for a mixed capacitated arc routing problem: A refuse collection application. *Eur. J. Oper. Res.*, 160, 139–153.

[35] M. Mourão & L.S. Pinto (2017) An updated annotated bibliography on arc routing problems. *Networks*, 70, 144–194.

[36] L. Muyldermans, D. Cattrysse & D.V. Oudheusden (2003) District design for arc-routing applications. *J. Oper. Res. Soc.*, 54, 1209–1221.

[37] J.B. Orlin (1993) A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 41, 338–350.

[38] I.H. Osman (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res.*, 41, 421–451.

[39] M. Polacek, K.F. Doerner, R.F. Hartl & V. Maniezzo (2008) A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *J. Heuristics*, 14, 405–423.

[40] C. Prins (2015) The capacitated arc routing problem: heuristics. In Corberán & Laporte (eds.), pp 131–157.

[41] D.G. Rossit, D. Vigo, F. Tohmé & M. Frutos (2019) Visual attractiveness in routing problems: a review. *Comput. Oper. Res.*, 103, 13–34.

[42] D.J. Rosenkrantz, R.E. Stearns & P.M. Lewis (1977) An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6, 563–581.

[43] P. Toth & D. Vigo (eds.) (2014) *Vehicle Routing: Problems, Methods and Applications*. Philadelphia, PA: SIAM.

[44] G. Ulusoy (1985) The fleet size and mix problem for capacitated arc routing. *Eur. J. Oper. Res.*, 22, 329–337.

[45] F.L. Usberti, P.M. França & A.L.M. França (2013) GRASP with evolutionary path-relinking for the capacitated arc routing problem. *Comput. Oper. Res.*, 40, 3206–3217.

[46] E.J. Willemse & J.W. Joubert (2016) Constructive heuristics for the mixed capacity arc routing problem under time restrictions with intermediate facilities. *Comput. Oper. Res.*, 68, 30–62.

[47] E.J. Willemse & J.W. Joubert (2016) Splitting procedures for the mixed capacitated arc routing problem under time restrictions with intermediate facilities. *Oper. Res. Lett.*, 44, 569–574.

[48] E.J. Willemse & J.W. Joubert (2019) Efficient local search strategies for the mixed capacitated arc routing problems under time restrictions with intermediate facilities. *Comput. Oper. Res.*, 105, 203–225.

[49] S. Wøhlk & G. Laporte (2018) A fast heuristic for large-scale capacitated arc routing problems. *J. Oper. Res. Soc.*, 69, 1877–1887.

Algorithm 3: Trip Selection

```
input : Number of required links m and trip length len[i] for
        i = 1, ..., m
Set N to ∞;
for i = 1 to m do
    Set k to 1, cu[0] to 0 and cu[1] to len[i];
    while cu[k] < m do
        Set cu[k + 1] to cu[k] + len[i+cu[k]];
        // Check if all required links have been served
        if cu[k + 1] ≥ m then
            Set cu[k + 1] to m;
        else
            // Check if we can serve more required links by
               backtracking
            for j = cu[k-1]+1 to cu[k]-1 do
                if j + len[i + j] > cu[k + 1] then
                    Set cu[k] to j and increase cu[k + 1] to j +
                      len[i + j];
                end
            end
        end
        Increase k by 1;
    end
    if cu[k] ≥ m then
        Set cu[k] to m;
    end
    if k < N then
        Set N to k and i* to i;
        for k = 1 to N do
            Set cu*[k] to cu[k];
        end
    end
end
output: Number of trips N, starting point i*,
        and collection of trips (represented by cu*[1] to cu*[N])
```

Algorithm 4: Initial Lower Bounds

input : mixed graph $G = (V, L)$, sets $L_R \subseteq L$, $I \subset V$,
demands q_ℓ, servicing times s_ℓ, traversal times t_ℓ,
vehicle capacity Q, deadline T, positive integer h

for $i = 1, \ldots, |L_R|$ **do**

let ℓ_i denote the ith closest required link from the depot;
let ℓ'_i denote the ith closest required link to the set I;
let ℓ''_i denote the ith closest required link from the set I;
let ℓ'''_i denote the ith closest required link to the depot via a node in I;

end

Set K to h and set α to $\max\left\{h, \left\lceil \sum_{\ell \in L_R} q_\ell / Q \right\rceil\right\}$;

repeat

Let $\tau = \sum_{\ell \in L_R} s_\ell + \sum_{i=1}^{K} t(\ell_i) + \sum_{i=1}^{\alpha-K} t'(\ell'_i) + \sum_{i=1}^{\alpha-K} t''(\ell''_i) + \sum_{i=1}^{K} t'''(\ell'''_i)$;
If $\tau/T > K$, then set K to $K + h$;
If $K > \alpha$, then set α to K;

until *no further increase in* K *is possible*;

output: Initial lower bounds K, α and τ.

Algorithm 5: Cutting-Plane Algorithm

input : mixed graph $G = (V, L)$, sets $L_R \subseteq L$, $I \subset V$, demands q_ℓ, vehicle capacity Q, deadline T, positive integer h, initial lower bounds K, α, τ

Construct the initial LP relaxation;
Solve the initial LP via primal simplex;
repeat
  Update τ;
  Set `improved` to *false*;
  if $\tau/T > K$ **then**
    Set `improved` to *true*;
    Set K to $K + h$ and update constraint (6);
    if $K > \alpha$ **then**
      Set α to K and update constraint (7);
    end
  else
    Call separation algorithms for constraints (11) and (12);
    if *any violated inequalities are found* **then**
      Set `improved` to *true*;
      Add the inequalities to the LP;
    end
  end
  Re-optimise the LP via dual simplex;
  Delete all cutting planes that have slack > 0.1;
until `improved` = false;
output: Final lower bounds K, α and τ.

City	Centre	Len 1	Len 2	Len 3	Len 4
Seoul	Arario Gallery Seoul	2229.5	3032.5	3567.5	4088.1
NewYork	RidgeWood	2969.0	3942.0	4892.6	5676.8
Istanbul	City Center AVM	1269.0	1839.0	2361.0	2811.0
Hanoi	National Cinema Center	1947.0	2811.0	3559.0	4299.0
London	MayFair	1898.0	2811.0	3516.0	4040.0

Table 1: Computation of initial squares.

City	$\|V\|$	$\|L\|$	$\|A\|$	$\|L_R\|$	$\|A_R\|$
Seoul	2446	3483	722	1760	353
	4972	7034	1186	3479	583
	7478	10625	1535	5374	725
	9936	14134	1817	7122	951
New York	2447	4072	2258	2057	1159
	4939	8334	5058	4207	2547
	7416	12630	7634	6285	3822
	9849	16877	10213	8373	5084
Istanbul	2490	3776	368	1925	176
	4961	7627	921	3745	464
	7432	11454	1501	5741	743
	9862	15219	2353	7696	1221
Hanoi	2356	3150	779	1570	377
	4918	6482	1558	3199	774
	7439	9951	2186	4977	1080
	9853	13222	2848	6630	1474
London	2437	3505	1779	1760	888
	4927	6950	2900	3537	1500
	7415	10296	3945	5034	1957
	9899	13508	4628	6674	2278

Table 2: Summary statistics for 20 graphs

City	$\|V\|$	K	α	Duration (hrs) mean	max	Time (s)
Seoul	2446	10	19	6.87	7.50	1.20
	4972	20	39	7.43	7.75	2.35
	7478	30	59	7.56	7.75	5.13
	9936	40	79	7.74	7.91	8.71
New York	2447	20	28	4.44	4.67	4.77
	4939	30	59	6.49	6.67	7.12
	7416	40	79	7.14	7.25	10.80
	9849	60	119	7.01	7.08	23.13
Istanbul	2490	10	20	6.80	7.50	1.62
	4961	20	39	7.05	7.33	4.93
	7432	30	60	7.54	7.75	6.76
	9862	40	79	7.68	7.83	11.94
Hanoi	2356	10	19	6.13	6.75	2.07
	4918	20	39	6.73	7.00	4.96
	7439	30	59	7.19	7.42	7.32
	9853	40	79	7.36	7.50	14.12
London	2437	10	19	6.77	7.41	1.48
	4927	20	39	7.31	7.67	3.19
	7415	30	59	7.17	7.33	9.65
	9899	40	79	7.37	7.50	13.70

Table 3: Upper bounding results under default scenario

City	$\|V\|$	K	α	τ	Ratio K	Ratio τ	Time (s)
Seoul	2446	10	16	63.4	1.000	0.923	65.54
	4972	20	34	137.3	1.000	0.924	67.67
	7478	30	52	207.8	1.000	0.916	71.50
	9936	40	70	283.0	1.000	0.914	85.59
New York	2447	10	20	79.2	0.500	0.891	63.40
	4939	30	41	176.5	1.000	0.906	65.44
	7416	40	60	259.8	1.000	0.909	68.83
	9849	50	81	365.3	0.833	0.868	82.12
Istanbul	2490	10	18	64.6	1.000	0.949	62.51
	4961	20	36	133.3	1.000	0.945	70.03
	7432	30	56	211.7	1.000	0.935	74.92
	9862	40	75	285.9	1.000	0.930	85.95
Hanoi	2356	10	15	57.4	1.000	0.937	62.41
	4918	20	31	124.6	1.000	0.925	76.81
	7439	30	48	195.9	1.000	0.908	68.97
	9853	40	64	268.0	1.000	0.911	74.66
London	2437	10	17	63.6	1.000	0.939	61.90
	4927	20	34	135.5	1.000	0.926	71.26
	7415	30	48	196.9	1.000	0.915	74.19
	9899	40	65	268.5	1.000	0.912	73.50

Table 4: Lower bounding results under default scenario