Citation for published version:
Wøhlk, S & Laporte, G 2018, 'A fast heuristic for large-scale capacitated arc routing problems', *Journal of the Operational Research Society*, vol. 69, no. 12, pp. 1877-1887. https://doi.org/10.1080/01605682.2017.1415648

DOI:
10.1080/01605682.2017.1415648

Publication date:
2018

Document Version
Peer reviewed version

Link to publication

A Fast Heuristic for Large-Scale Capacitated Arc Routing Problems

Sanne Wøhlk[1] and Gilbert Laporte[2]

[1] CORAL - Cluster for Operations Research and Logistics, Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark. sanw@econ.au.dk

[2]GERAD and Canada Research Chair in Distribution Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7. gilbert.laporte@cirrelt.ca

Abstract

The purpose of this paper is to develop a fast heuristic called FastCARP for the solution of large-scale capacitated arc routing problems, with or without duration constraints. This study is motivated by a waste collection problem in Denmark. After a preprocessing phase, FastCARP creates a giant tour, partitions the graph into districts, and construct routes within each district. It then iteratively merges and splits adjacent districts and reoptimises the routes. The heuristic was tested on 264 benchmark instances containing up to 11,640 nodes, 12,675 edges, 8,581 required edges, and 323 vehicles. FastCARP was compared with an alternative heuristic called Base. On small graphs, it was better but slower than Base. On larger graphs, it was much faster and only slightly worse than Base in terms of solution quality.

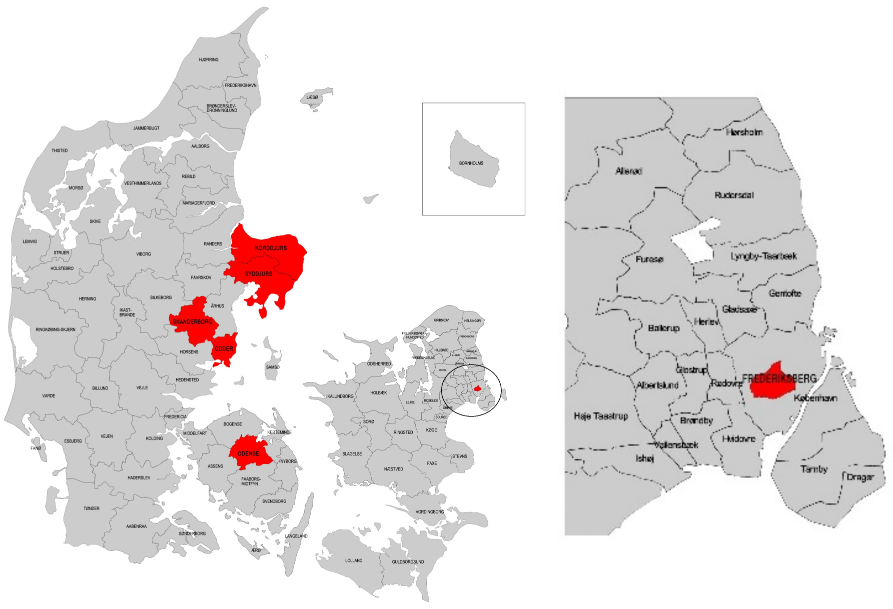
Keywords Arc routing, Districts, Heuristics, Waste collection, Denmark.

1 Introduction

The purpose of this paper is to develop a fast constructive heuristic called FastCARP for a large-scale Capacitated Arc Routing Problem (CARP) arising in waste collection operations in Denmark, but also in several other settings.

The CARP, introduced by (Golden and Wong, 1981), is defined as follows. Let $G = (N, E)$ be an undirected connected graph, where N is the set of nodes and E is the set of edges. A subset E_R, of the edges require service. Each such edge $e \in E_R$ has a demand q_e. The traversal cost of edge e is denoted by c_e and is independent of whether the edge is being serviced or merely deadheaded. A special depot node holds a fleet of identical vehicles of capacity W. The CARP amounts to identifying a closed route R_k for each vehicle k such that 1) every route starts and ends in a common depot, 2) every required edge is serviced by one of the vehicles, 3) the vehicle capacities are not exceeded by the sum of the demands q_e on their route, and 4) the total routing cost is minimised. A mathematical description of the problem can be found in (Golden and Wong, 1981). See (Muyldermans and Pang, 2015) and (Kiilerich and Wøhlk, 2017) for a discussion of various CARP variations, as well as (Ahr and Reinelt, 2015) and (Prins, 2015) for a recent review of lower bounds and algorithms for the CARP.

In our application, we also impose a route duration constraint. For this purpose, let s_{ij} (t_{ij}) be the service (deadheading) time for edge (i, j) and let T be the duration limit of a route. Then every route R_k starts at time zero and must be completed by time T.



Denmark. *Zoom on Copenhagen.*

Figure 1: The six counties providing the data.

1.1 The Danish Application

In Denmark, waste collection falls under the responsibility of 98 counties. Our study concentrates on six of these counties (Figure 1) of which are two urban (Frederiksberg (F) and Odense (O)), two are semi-urban (Skanderborg and Odder (K)), and two are rural (North (N) and South (S) Djurs). Figure 2 depicts the graphs of these counties.

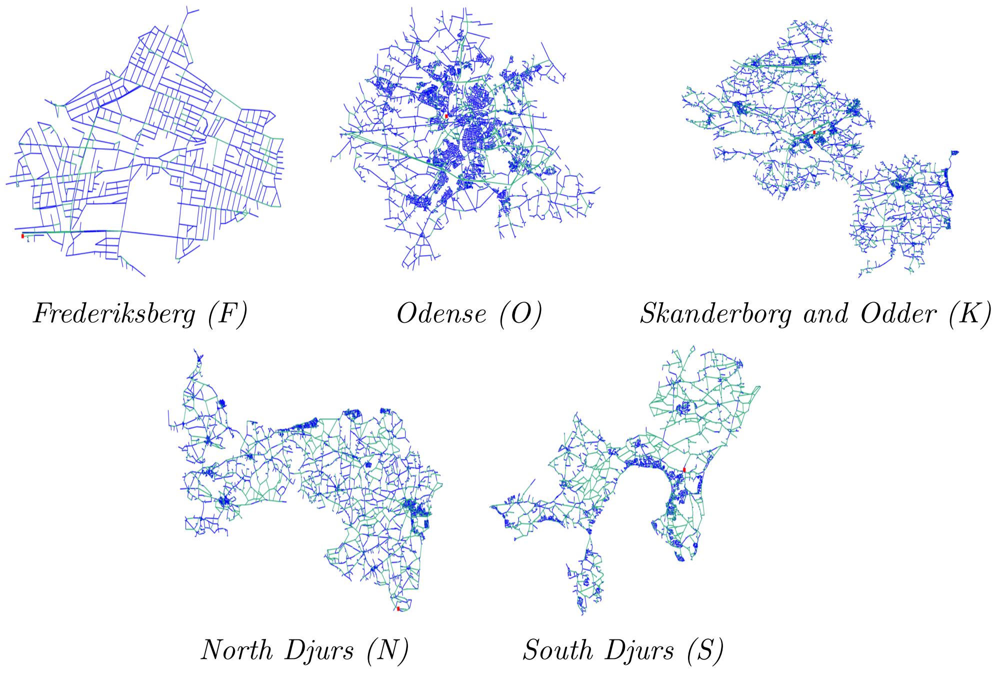


Figure 2: The five data areas. Blue edges are required and green edges are non-required. The depot is marked by a red square.

The study of waste collection as an operational research topic dates back to the 1970s (Clark and Gillean, 1977). Waste collection systems are designed and analysed at the strategic level (Adamides et al., 2009). They involve the location of treatment facilities, of transfer points, and of landfills (Khan, 1987; Kim and Lee, 2015). At the tactical level, decisions must be made on the type and amount of waste that should be sorted, and collection districts must also be designed (Muyldermans et al., 2003; Mourão et al., 2009; Butsch et al., 2014). Finally, at the operational level, routes are planned (Mourão and Amado, 2005; Ghiani et al., 2015) and one must also decide whether different types of waste should be co-collected (Muyldermans and Pang, 2010). Finally, collection schedules must be constructed. For further information on waste collection systems, we refer the reader to (Letcher and Vallero, 2011), and (Ghiani et al., 2015).

Our study focuses on the operational level. We consider route planning for kerbside collection in the six counties mentioned above, both with and without route duration constraints. To give the reader an idea of the size of the operations, in Odense approximately 57 tons of waste are collected annually by kerbside mode from a total of 95,199 households spread over 305 km^2 (2015 statistics). Because each county is responsible for planning its own waste collection system, there are some where citizens are free to choose from a set of collection intervals, and others where the collection days are imposed by the authority. In some counties, only general waste is kerbside collected, whereas up to seven different waste types are kerbside collected in others: general waste, organic waste, plastic, metal, glass, paper, and cardboard. Furthermore, some counties handle the waste collection operations themselves, whereas others outsource the operations to a logistics provider. This is further discussed in (Kiilerich and Wøhlk, 2017).

In the course of this work, we collected information from the planners in the six Danish counties and we found that their fleet mix and their constraints vary slightly across the counties. In order to make our contribution as generic as possible, we have made some simplifications to the problems faced by the six counties and we work with representative figures. Namely, we work with a single vehicle type, only one type of waste, and a common maximal route duration of 7.5 hours.

The instances under consideration are highly variable in size, but the majority are of large scale, attaining up to 11,640 nodes, 12,675 edges, 8,581 required edges, and 323 vehicles.

1.2 Available Algorithms for the CARP

There are three types of algorithms for solving problems like the CARP: exact algorithms, metaheuristics, and classical (simple) heuristics.

The exact algorithms can solve the problem to optimality (Belenguer et al., 2015). However, because exact algorithms are only able to solve instances of limited sizes, heuristics are typically used in practice.

Due to their ability to obtain high quality solutions and their high degree of flexibility regarding incorporation of side constraints, metaheuristics are usually preferred. The first metaheuristic for the CARP was a simulated annealing heuristic used for the solution of a winter gritting application (Eglese, 1994). Since then, several metaheuristics have been suggested for the CARP, including tabu search (Hertz et al., 2000; Brandão and Eglese, 2008), variable neighbourhood search (Polacek et al., 2008), guided local search (Muyldermans, 2003; Usberti et al., 2013), and many more. The most recent metaheuristic contribution for the CARP is described in (Vidal, 2017), in which neighbourhood evaluations are performed very efficiently. See (Prins, 2015) for an overview. Typically, metaheuristics require a very large number of iterations to obtain high quality solutions. As a result, they are time consuming for large instances and impractical for very large instances. They are usually tested on benchmark instances, the largest being those of (Li and Eglese, 1996) with up to 140 nodes, 190 edges, and 190 required edges and of (Brandão and Eglese, 2008) with up to 255 nodes, 375 edges, and 375 required edges.

When solutions are needed faster than what the metaheuristics can deliver, simpler constructive heuristics are preferred. Several constructive heuristics have been suggested for the CARP, the classical ones being the Augment-Merge algorithm of (Golden and Wong, 1981) and the Path-Scanning algorithm of (Golden et al., 1983) and more recently, the Double Outer-Scan heuristic of (Wøhlk, 2005). Recently, (Zbib, 2017) presented results for the large scale instances presented in (Kiilerich and Wøhlk, 2017) using a large number of path scanning variations. See (Prins, 2015) for an overview of constructive heuristics for the CARP.

Given the fact that we are interested in the solution of large-scale instances, the time it would take to run a metaheuristic to completion would be prohibitive. Since our aim is to develop a fast heuristic for large instances, we cannot reasonably compare it with any of the available metaheuristics. We have therefore chosen to compare our heuristic, FastCARP, with the most natural constructive heuristic in which a giant route is first created and subsequently partitioned into several vehicle routes. We refer to this benchmark heuristic as Base. We also compare our results to the path scanning results (PS) obtained by (Zbib, 2017).

1.3 Contribution and Organization of the Paper

The contribution of this paper is to present a fast heuristic for solving large CARPs. The heuristic is sufficiently flexible to easily incorporate additional constraints, such as route duration limits. This paper is the first to provide solutions for the large CARP benchmark instances provided in (Kiilerich and Wøhlk, 2017).

The remainder of this paper is organised as follows. In Section 2, we describe our heuristic. In Section 3, we present our computational results, and in Section 4, we provide our conclusions.

2 Description of FastCARP

The overall idea of FastCARP is quite simple. After some preprocessing, an giant tour R is created using the algorithm of (Frederickson, 1979), and is partitioned into a number of districts, where the term district is used to describe a partial giant tour, often requiring the use of several vehicles and rarely possessing traditional district attributes such as compactness and convexity. The districts are stored in an ordered list $\mathcal{D} = (D_0, D_1, \ldots, D_{|\mathcal{D}|-1})$. We then iterate through the list in a cyclic manner and consider two adjacent districts at a time. The two districts are merged into one, which is then optimised and split into two districts again. During this process, which is repeated until an overall time limit is reached, we are careful about maintaining the ordering of the districts. Throughout the algorithm, we keep track of the best known feasible solution. The heuristic is outlined in Algorithm 1, whereas the district optimisation is detailed in Algorithm 2. With respect to Base, FastCARP takes advantage of the natural geographical decomposition of an instance into districts. It considers fewer and more meaningful cutting moves than Base. Consequently, as our results will show, it tends to be faster and usually better than Base.

Algorithm 1 FastCARP

```
$G \leftarrow$ Preprocessing
$R \leftarrow$ CreateGiantTour
$\mathcal{D} = (D_0, D_1, \dots, D_{|\mathcal{D}|-1}) \leftarrow$ CreateDistricts($R$)
Set $U = \emptyset$
$k_1 = 0,\ k_2 = 1$
while Runtime < RunTimeLimit do
    $D \leftarrow$ MergeDistricts($D_{k_1}, D_{k_2}$)
    $D, U \leftarrow$ OptimiseDistrict($D, U$)
    $D_{k_1}, D_{k_2} \leftarrow$ SeparateDistrict($D$)
    $k_1 \leftarrow k_1 + 1(\text{mod}|\mathcal{D}|),\ k_2 \leftarrow k_2 + 1(\text{mod}|\mathcal{D}|)$
end while
```

The OptimiseDistrict procedure uses and modifies some of the procedures developed by (Hertz et al., 1999) for the Rural Postman Problem (RPP) and by (Hertz et al., 2000) for the CARP (the latter algorithm is called Carpet), but it contains some important differences as described below. OptimiseDistrict takes as input a district containing a list of routes. In the main loop, the procedure Paste concatenates all routes into a giant tour R. In Switch (Hertz et al., 1999), a node v occurring multiple times in R is identified and the subsequences of the route between any two succeeding occurrences of v are reversed, after which v is added to the set U of used nodes. The Shorten procedure (Hertz et al., 1999) is then applied to the resulting giant tour which is again partitioned into feasible routes by the

Split procedure. Finally, the individual routes are optimised by Shorten if possible.

2.1 Preprocessing and Creation of a Giant Tour

We first consider the part of FastCARP (Algorithm 1) that comes before the main loop. In the preprocessing, we merge sequences of edges that end in a dead-end edge and which have no side edges. We also tried merging other sequences of edges without a side edge, but this resulted in worse solutions due to the limitations of the Split procedure.

In order to create the initial giant tour, we apply the algorithm of (Frederickson, 1979) for the creation of an RPP solution, with a slight modification. The original procedure is as follows. First, a minimum cost spanning tree is determined in a graph where each node represents a connected component with respect to required edges in G and costs are the minimum cost between edges in the components. After adding the edges of this minimum spanning tree to G, the set S of odd-degree nodes with respect to required edges and the edges of the tree is identified. (Frederickson, 1979) computes a minimum cost perfect matching among the nodes in S. However, according to (Wøhlk and Laporte, 2017) it is not practical to seek an optimal solution to the matching problem for instances as large as those considered in this paper. We therefore use the heuristic Sum of that paper to solve the matching problem. The heuristic of (Wøhlk and Laporte, 2017) yields solutions that are on average 13.6 percent above the optimum. After adding the edges of the matching to G, the graph induced by the required edges, the spanning tree edges, and the matching edges is Eulerian and the algorithm by (Hierholzer, 1873) is applied to find a giant tour in G.

2.2 Creation of Districts

The next step in the algorithm is the procedure CreateDistricts which partitions the giant tour into an ordered sequence of districts $\mathcal{D} = (D_0, D_1, \ldots, D_{|\mathcal{D}|-1})$. This is essentially done by the CutMax procedure (described in Section 2.3) where the time duration constraints are ignored and the vehicle capacity is replaced by βW, where $\beta > 1$ is determined during tuning. If the last district is sufficiently small (corresponding to a demand of at most $1.1W$), it is immediately concatenated with the second to last. We have tested the following three ways of determining β. Let $K = \lceil \sum_{e \in E} q_e / W \rceil$ be the estimated number of routes in the solution (and indeed a lower bound on the number of routes). 1) Then the first approach is to aim for a fixed (small) number of routes in each district (e.g. $\beta = 3$), which stabilises the time for OptimiseDistrict and allows for more iterations than the other two approaches. 2) The second is to set $\beta = \lceil \sqrt{K} \rceil$ and thereby aim for the same number of districts as the number of routes in each district. This tends to yield larger districts and thereby more options for optimisation, but also longer run times. 3) Finally, we have tried to use a fixed number of districts. However, due to the varying sizes of the instances, this was discarded. Indeed, our tuning experiments showed that setting $\beta = \lceil \sqrt{K} \rceil$ performed best overall and this is the value that we used in the final implementation of the algorithm. This value provides a balance between flexibility and speed, in the sense that it results in districts that are large enough to provide the flexibility needed to optimise them thoroughly, but are small enough not to be too time consuming to optimise, thereby allowing the algorithm to cycle through the districts multiple times.

2.3 Optimisation of the Districts

The main loop of OptimiseDistrict is repeated ψ times, where $\psi = 25$ have been determined during our tuning phase. At each iteration of the main loop, a new node $v \in N \setminus U$ is selected in Switch (Hertz et al., 1999). If no such node can be found, the set U is cleared ($U = \emptyset$) and the search is reiterated. We have also tried to clear U whenever OptimiseDistrict is entered, but this turned out to be less successful. We have tested the following four methods to select $v \in N \setminus U$. 1) Consider the nodes $v \in N \setminus U$ in the order in which they occur in the list of nodes as read from the data file, which is an arbitrary order, and use the first node that occurs more than once in the giant tour under consideration. 2) Consider the nodes $v \in N \setminus U$ in the order in which they appear on the giant tour and select the first one that occurs more than once. 3) Select the node $v \in N \setminus U$ that occurs most often on the giant tour and break ties arbitrarily. 4) Select the node $v \in N \setminus U$ with the most serviced edges between any two succeeding occurrences. Tuning indicates that method 1 generally performs best (closely followed by method 4) and is the one used in our final implementation. We believe that the superiority of this rule comes from the fast that it does not favour any of the extreme situations of the selection nodes (many occurrences, long distance between them, early on the route), but rather uses nodes with a mix of these attributes. After identifying the node $v \in N \setminus U$, SWITCH reverses the orientation of the cycles between any two successive occurrences of v.

Algorithm 2 OPTIMISE DISTRICT

```
procedure OptimiseDistrict($D = (R_1, R_2, \ldots)$)
    for $i = 1$ to $\psi$ do
        $R \leftarrow$ Paste($R_1, R_2, \ldots$)
        $R, U \leftarrow$ Switch($R, U$)
        $R \leftarrow$ Shorten($R$)
        $(R_1, R_2, \ldots) \leftarrow$ Split($R$)
        $R_k \leftarrow$ Shorten($R_k$) $\forall\ k$
    end for
end procedure
```

An essential part of OPTIMISEDISTRICT is the SPLIT procedure, where the giant tour of a district is partitioned into routes that are feasible with respect to both capacity and duration constraints. We have tested five variants of it in our tuning phase. 1) The first variant is the CUT procedure of the CARPET algorithm (Hertz et al., 2000), which searches for a feasible point to end the first route yielding the smallest additional cost, splits the giant tour at that point, and reiterates the process. The advantage of CUT is that it is fast. However, for large-scale instances, it results in poor solutions due to a tendency to split the tour very early, which results in significantly more routes than necessary. 2) The second version is the SPLIT procedure (without flipping), presented in (Prins et al., 2009), which splits the giant tour in an optimal fashion using a shortest path algorithm. This version is clearly the best of the four, but it is far too time consuming to be used within our algorithm for large-scale instances. 3) The third version, CUTMAX, splits the route as late as possible while ensuring that the first route is feasible, and repeats this operation. Its advantage is that it tends to create long routes, while its drawback is that it often does not select good points where to cut the giant tour. This is remedied in the last two variants. 4) CutLimit, which is a simple combination of CutMax and Cut, works as follows. Let α be a number between zero and one, determined in the tuning phase. Scanning from the beginning of the giant tour, the procedure first identifies a safety point as the latest possible node where to split the tour so that the demand serviced on the first route is at most αW and the route duration is at most T. Between this point and the latest feasible split point, the rule defined in Cut is used to identify a good splitting point. If no such point is found, the tour is split at the safety point. This process is then repeated for the next route. This version is as fast as CutMax, but performs better. 5) The fifth and final version, CutLimit$'$, combines features from CutLimit and Split. Here, for each possible start required edge of the giant tour, we consider the α' latest points where to cut the route while feasible. Using a shortest path algorithm, we then determine the best combination of splitting points. In order to keep the computation time low, no flipping is performed. This is the version that proved the most successful for large instances and it is the one we have used in our final implementation.

The idea in Shorten, which is originally developed by (Hertz et al., 1999), is to follow the route from a starting point and every time an edge is being serviced, postpone the service of that edge to a later traversal of the same edge. This results in sequences of deadheading edges, which can be replaced by a shortest path. In the original description, the process stops when demand of an edge cannot be postponed, but then the whole process is restarted using another starting point, thereby viewing the tour as one large cycle. In our implementation, we consider the tour in a linear fashion. We therefore only start from the beginning of the route, but after we encounter a required edge were we cannot postpone the demand, we continue the postponement and shortening from that point. This causes the postponement in our procedure to differ from the original implementation.

2.4 Illustration of FastCARP

We illustrate FastCARP in Figure 3, where the top shows a simplified version of the giant tour of three districts in the starting state and the middle (bottom) shows the situation after the first (second) iteration of the main loop in Algorithm 1.

Consider first the starting state. District 1 (dashed) starts at the leftmost point, continues through point v along the dashed line until point x. At x, district 2 (solid line) takes over, continues through v and u to y. District 3 (dotted) ranges from y through u to the rightmost point.

In FastCARP, we merge two adjacent districts, optimise them jointly and separate them again. Without this merge, the starting situation in our illustration would not provide opportunities for further optimisation.

In the first iteration of Algorithm 1, districts 1 and 2 are concatenated at v.

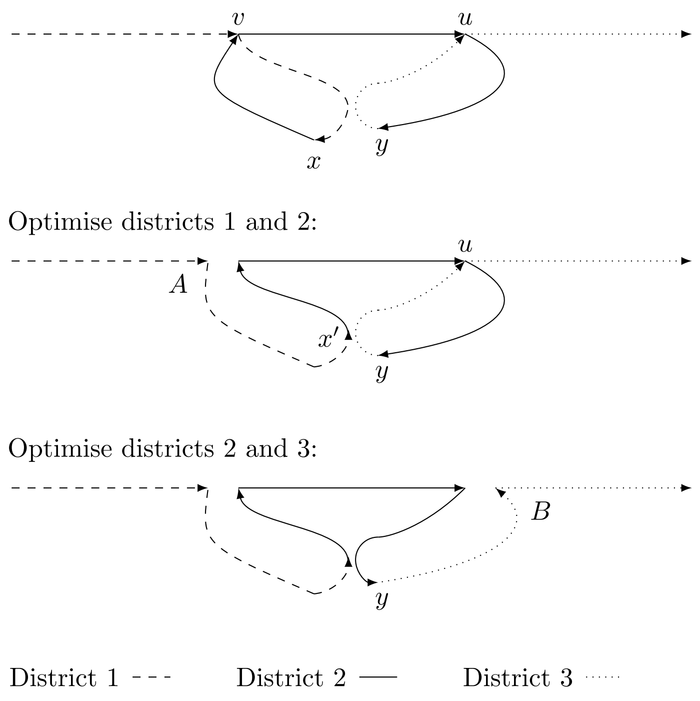


Figure 3: Illustration of two iterations of FastCARP

This provides the option of selecting v as switch node, which causes the cycle through v and x to change orientation. SHORTEN is then applied and causes a route reduction near v. This is marked with an A in the middle picture. Finally, the union of the two districts is split into routes using CUTLIMIT$'$ and the joint district is separated into two by assigning the first half of the routes to district 1 and the remaining routes to district two. This causes the separation point between the districts to move from x to x' as shown in the middle picture.

In the second iteration, districts 2 and 3 are merged at node u. Now, u can be selected as switch node and the cycle through u and y changes orientation. Applying SHORTEN leads to an improvement near u (marked with B in the button picture). CUTLIMIT$'$ is used to split the giant tour into routes and the joint district is separated into two again. This time the separation point remains unchanged at point y.

Besides the improvements directly illustrated in Figure 3, district 2 is now in a good position for further optimisation in subsequent iterations.

3 Computational Results

We have tested our algorithm on the 264 CARP benchmark instances of (Kiilerich and Wøhlk, 2017) which consist of 88 graphs, each combined with three different vehicle capacities. The largest instance contains 11,640 nodes, 12,675 edges, and 8,581 required edges. The number of vehicles needed to service all demand ranges from 2 to 323, and using the minimum number of vehicles results in three to 440 required edges being serviced by each vehicle. The instances are available at http://www.optimization.dk/CARP/.

For the tests with inclusion of duration constraints, we need a common travel speed. We have chosen to use an estimated average speed, which will by default overestimate speed in urban regions and underestimate it in more rural areas. Likewise, after consulting one of the logistics providers, we have estimated the time for empting waste bins. The true time depends on the number of workers assigned to each vehicle and an analysis of this aspect is beyond the scope of this paper. We have used the following settings, where the numbers of bins to be serviced on each edge are provided in the graph files, and the distances are given in meters:

travel speed:	60 km/h,
additional service time:	1.5 minute per bin,
route duration limit:	7.5 hours.

As mentioned in Section 1, we used the Base algorithm as a benchmark for the assessment of FastCARP. It was implemented by successively executing Preprocessing, CreateEulerTour, and Split.

Tuning of FastCARP was performed on 15 of the 264 CARP instances. In total, approximately 200 combinations of parameters and algorithmic choices were tested in the tuning phase, leading to the final choices described in Section 2.

Both algorithms were implemented in C^{++} in MS Visual Studio Professional 2015 and executed on an Intel Xeon CPU with 12 cores running at 3.5 GHz and 64 GBs RAM. They were executed sequentially, i.e. without taking advantage of the multiple cores, and the programs were run simultaneously on two threads. Since our aim was to design a fast algorithm, we only allowed one minute of runtime per 1,000 nodes in the graph. This means that the largest instances are solved within less than 12 minutes, excluding the time needed to read the graph and create the shortest path matrix. The run time for the BASE algorithm is not limited since the algorithm stops after SPLIT.

For the CARP without duration constraints, we also use the results presented in (Zbib, 2017) as benchmark. Those results represent the best solutions obtained by five runs of each of 21 Path Scanning algorithms (PS) based on seven different evaluation criteria and three different degrees of randomization. Those results were obtained on a VMware virtual machine with the following specs: Intel(R) Xeon(R) CPU E5-2680 0 at 2.70GHz with 64GB of vRAM. The run times reported are the aggregated time for the 105 algorithm runs for each instance.

We first illustrate our algorithm on an example instance, S12_g-3, which covers a rural area with a few villages. The graph contains 755 nodes and 866 edges, of which 407 are required. When this instance is solved, four routes are needed, each servicing on average 102 required edges. Figure 4 depicts the routes traversed and the edges services by each of the four vehicles in the solution produced by FASTCARP. As can be seen from these figures, the solution is quite visually appealing. This is often, but not always, the case. To illustrate, Figure 5 shows the edges services by each of four vehicles in instance S7_g-4. The graph contains 2,099 nodes and 2,591 edges of which

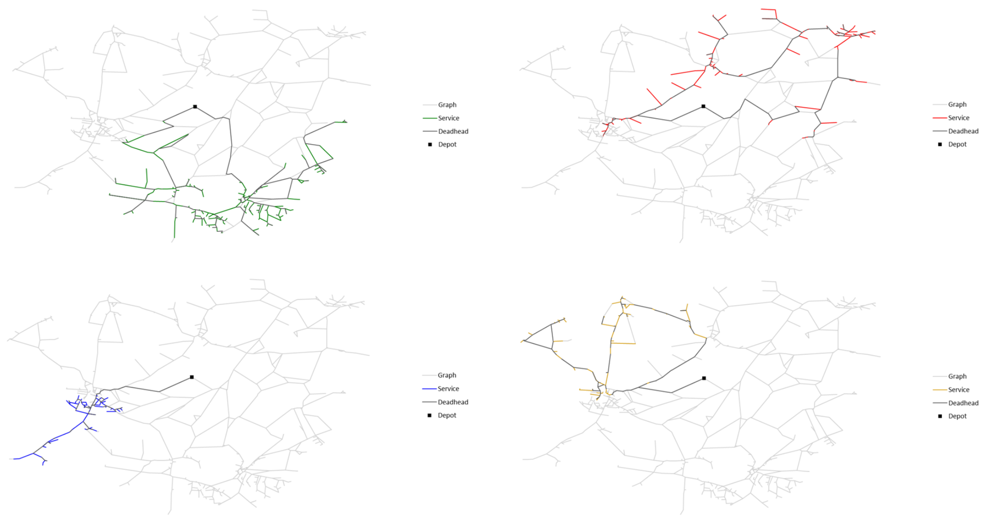


Figure 4: The four routes of instance S12_g-3 solved by FastCARP.

864 are required. In this solution, it can be seen that the red and the gold routes are intertwined.

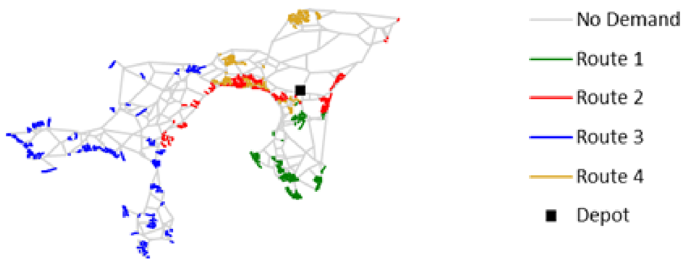
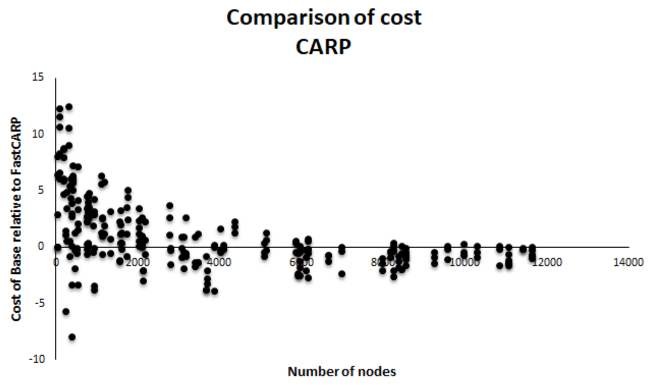


Figure 5: Example of solution. Instance S7_g-4.

Due to the very large number of instances we have solved, we only show detailed results for a limited number of them and report summarised results over all 264 instances. For future comparison, we provide detailed results at http://www.optimization.dk/CARP/. Figure 6 shows the cost obtained by



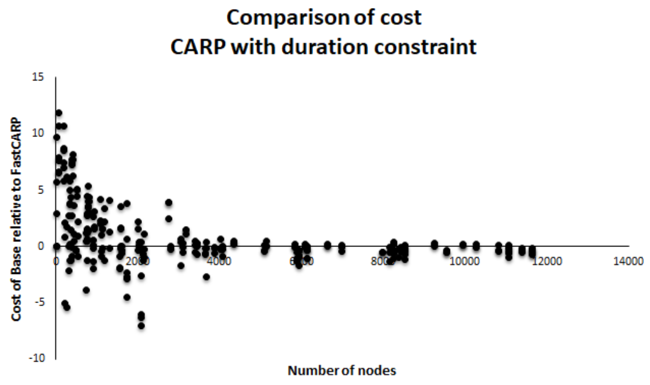


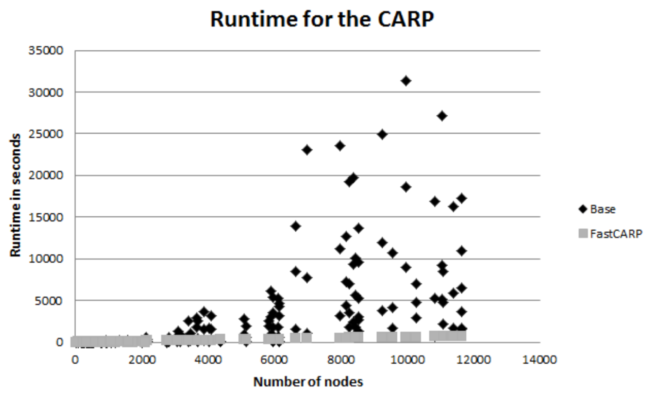
Figure 6: Relative cost of FastCARP compared to Base for CARP and CARP with duration constraint. Positive values indicate that FastCARP obtained better cost.

	CARP			CARP with duration constraint	
	Base	PS	FastCARP	Base	FastCARP
Average % above best	1.6	19.1	0.6	1.3	0.4
Max % above best	12.5	41.8	8.7	11.8	7.6
Total number	264	264	264	264	264
Number of best	125	1	141	133	135
Average number of routes	34.5	-	33.6	51.9	51.1
Max number of routes	185	-	185	323	230

Table 1: Summary of the computational results over all 264 instances. Information regarding individual routes are not available for PS.

FastCARP relative to the one obtained by Base for all instances solved as a standard CARP (top) and as a CARP with duration constraints (bottom). It is computed as $100 \times \frac{C(\textsc{Base})-C(\textsc{FastCARP})}{C(\textsc{FastCARP})}$. Hence positive values indicate that FastCARP is better and negative values mean that Base is better. Table 1 provides a summary of the results. Note that in the figure, some of the results of the Base algorithm are hidden behind those of FastCARP. For instances with up to about 3,000 nodes, FastCARP generally performs better than Base, whereas the Base algorithm yields slightly better results for the larger graphs. However, as can be seen from Figure 7, Base spends significantly more time for the large graphs. For instance, it takes more than eight hours to solve instance O1_p-4 with 9,957 nodes, while only about ten minutes are allowed. This large run time is due to the very high number of required edges per route in this instance. For this instance, the cost obtained by FastCARP in ten minutes is only 0.7% higher than that yielded by the Base algorithm. From Table 1 is is clear that PS is not competitive on these large graphs, even when the best over many variations is considered.

As is evident from Table 1, FastCARP exhibits a better average performance than Base, and in the light of the runtime, we conclude that FastCARP is indeed a fast algorithm for finding quite good solutions for the CARP.



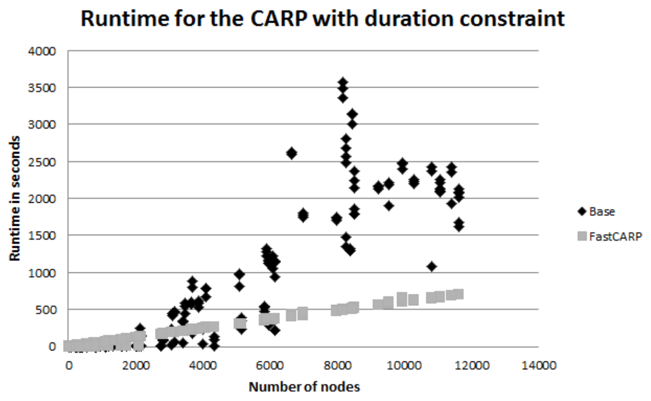


Figure 7: Run time for FastCARP and the base algorithm for CARP and CARP with duration constraint.

	Instance information				BASE				PS			FASTCARP				
Instance	$\|N\|$	$\|E\|$	$\|E_R\|$	% N	Cost	%	$\|\mathcal{R}\|$	Time	Cost	%	Time	Cost	%	$\|\mathcal{D}\|$	$\|\mathcal{R}\|$	Time
F1_g-4	812	1124	780	1.0	804877	4.8	118	0.2	843418	9.8	9.0	768209	-	10	113	48.7
F1_g-6	812	1124	780	1.0	491445	3.5	65	0.3	530687	11.8	6.5	474809	-	7	63	48.8
F1_p-2	795	1106	728	1.1	272287	4.0	27	1.3	287581	9.9	4.2	261743	-	5	27	47.7
F1_p-6	795	1106	728	1.1	163009	0.4	9	9.3	182656	12.5	3.3	162376	-	2	8	48.0
F9_g-4	788	1096	686	1.3	751015	4.6	109	0.2	795868	10.8	8.0	718145	-	9	105	47.3
F9_g-6	788	1096	686	1.3	458289	3.1	60	0.4	496164	11.7	5.9	444369	-	7	59	47.3
K1_g-2	11640	12675	8566	9.3	6453427	-	150	1564.2	7549094	17.0	1386.5	6501210	0.7	11	147	700.2
K1_g-6	11640	12675	8566	9.3	3711945	-	44	17254.9	4886345	31.6	1131.0	3739724	0.7	6	42	698.5
K2_g-2	11636	12671	8563	9.3	6248942	-	146	1730.6	7434491	19.0	1568.5	6249733	0.0	11	143	699.5
K2_g-4	11636	12671	8563	9.3	4401690	-	74	6471.6	5629112	27.9	1430.6	4434203	0.7	8	71	701.0
K5_g-2	11405	12435	8267	9.3	6029327	-	140	1678.7	7136324	18.4	1588.3	6031579	0.0	11	139	685.6
K5_g-6	11405	12435	8267	9.3	3571768	-	42	16328.5	4707990	31.8	1441.1	3578627	0.2	6	39	685.5
N1_g-2	8537	9725	6339	7.3	6472575	-	90	1262.6	7299853	12.8	716.6	6476448	0.1	9	89	513.5
N1_g-6	8537	9725	6339	7.3	3180997	-	26	13692.1	4089441	28.6	599.9	3207125	0.8	5	26	514.0
N2_g-3	8537	9725	6339	7.3	4716390	-	57	3029.4	5589920	18.5	750.0	4749044	0.7	7	57	512.5
N2_g-5	8537	9725	6339	7.3	3436348	-	31	9566.8	4337055	26.2	692.7	3492355	1.6	5	31	516.5
N5_g-2	8444	9631	6165	7.3	5914240	0.1	82	1854.0	6723954	13.8	776.2	5908863	-	8	80	508.1
N5_g-5	8444	9631	6165	7.3	3325667	-	30	10063.5	4220216	26.9	707.9	3392187	2.0	5	29	509.7
O1_g-4	10283	11863	8581	2.8	3282866	0.1	133	2896.3	4173408	27.3	1357.1	3278666	-	11	133	619.1
O1_g-6	10283	11863	8581	2.8	2700609	-	78	7035.4	3596504	33.2	1274.0	2724848	0.9	8	77	618.4
O1_p-2	9957	11492	8220	3.1	2499283	-	70	8944.0	3465916	38.7	1217.4	2509047	0.4	8	68	597.9
O1_p-4	9957	11492	8220	3.1	2179060	-	36	31417	3089185	41.8	1163.1	2194629	0.7	5	33	600.7
O6_g-2	9563	11073	7831	2.6	3534430	0.1	158	1670.5	4345961	23.1	1217.5	3531246	-	11	156	574.7
O6_g-6	9563	11073	7831	2.6	2273493	-	46	10656.6	3100213	36.4	1070.1	2276829	0.1	6	43	575.4
S1_g-1	6149	7110	3797	3.7	3651318	0.7	138	78.1	3987605	10.0	494.4	3624502	-	11	137	369.3
S1_g-6	6149	7110	3797	3.7	1470679	-	21	4212.5	1886632	28.3	416.3	1478193	0.5	4	19	370.4
S2_g-1	6149	7110	3797	3.7	3480732	0.6	128	85.7	3822136	10.5	493.4	3459811	-	10	127	369.3
S2_g-6	6149	7110	3797	3.7	1439140	-	19	4685.6	1851578	28.7	417.6	1478328	2.7	4	19	369.2
S5_g-2	6104	7063	3732	3.8	2191441	-	60	684.5	2588801	18.1	437.5	2197632	0.3	7	59	367.0
S5_g-6	6104	7063	3732	3.8	1413134	-	18	5238	1841206	30.3	410.3	1443084	2.1	4	17	367.3

Table 2: Computational results for CARP.

Instance	Instance information			BASE with duration constraints				FASTCARP with duration constraints				
	$\|N\|$	$\|E\|$	$\|E_R\|$	Cost	%	$\|\mathcal{R}\|$	Time	Cost	%	$\|\mathcal{D}\|$	$\|\mathcal{R}\|$	Time
F1_g-4	812	1124	780	813061	4.4	119	0.2	778756	-	10	114	48.7
F1_g-6	812	1124	780	542632	3.0	74	0.3	526983	-	7	72	48.7
F1_p-2	795	1106	728	275511	4.0	28	1.2	264874	-	5	28	47.7
F1_p-6	795	1106	728	210221	1.3	17	2.4	207434	-	2	17	47.9
F9_g-4	788	1096	686	751015	5.3	109	0.2	713085	-	9	103	47.3
F9_g-6	788	1096	686	464048	3.6	62	0.3	448126	-	7	59	47.3
K1_g-2	11640	12675	8566	6533842	-	151	1625.4	6577393	0.7	11	150	698.9
K1_g-6	11640	12675	8566	5878564	-	118	2088.0	5888625	0.2	6	117	702.8
K2_g-2	11636	12671	8563	6519326	-	151	1670.5	6536208	0.3	11	148	700.4
K2_g-4	11636	12671	8563	5879529	-	119	2121.2	5897610	0.3	8	118	699.8
K5_g-2	11405	12435	8267	6097035	-	142	1936.6	6106502	0.2	11	141	685.6
K5_g-6	11405	12435	8267	5442346	-	110	2356.7	5459343	0.3	6	110	687.3
N1_g-2	8537	9725	6339	6867118	0.1	95	1791.5	6859638	-	9	94	514.0
N1_g-6	8537	9725	6339	6554441	-	84	1795.8	6563203	0.1	5	84	519.9
N2_g-3	8537	9725	6339	6577542	-	85	2374.0	6626181	0.7	7	86	518.6
N2_g-5	8537	9725	6339	6549203	-	84	2139.6	6622589	1.1	5	84	535.5
N5_g-2	8444	9631	6165	6385417	-	88	3142.7	6410597	0.4	8	86	507.7
N5_g-5	8444	9631	6165	6091088	-	78	3006.2	6103146	0.2	5	77	518.4
O1_g-4	10283	11863	8581	4387039	-	232	2198.5	4391082	0.1	11	230	618.0
O1_g-6	10283	11863	8581	4357878	0.2	230	2229.4	4348033	-	8	226	623.7
O1_p-2	9957	11492	8220	3999125	0.1	196	2472.2	3994262	-	8	195	601.9
O1_p-4	9957	11492	8220	3999125	0.2	196	2476.1	3992181	-	5	196	651.6
O6_g-2	9563	11073	7831	3814083	-	175	1905.4	3833850	0.5	11	178	574.7
O6_g-6	9563	11073	7831	3813263	-	175	2179.3	3825362	0.3	6	176	595.5
S1_g-1	6149	7110	3797	3654156	-	138	220.2	3658287	0.1	11	138	370.5
S1_g-6	6149	7110	3797	2280897	-	60	1151.6	2289331	0.4	4	59	370.3
S2_g-1	6149	7110	3797	3485149	0.1	128	213.0	3482831	-	10	129	371.0
S2_g-6	6149	7110	3797	2278914	-	60	1154.5	2302618	1.0	4	58	369.5
S5_g-2	6104	7063	3732	2255075	0.2	61	1120.9	2250277	-	7	60	367.4
S5_g-6	6104	7063	3732	2167317	-	54	1226.5	2168346	0.0	4	52	372.8

Table 3: Computational results for the CARP with duration constraints.

Table 2 provides detailed results for the three largest instances in each of the five sets with each their smallest and largest vehicle for the CARP, and Table 3 gives the same results for CARP with duration constraints. The naming convention for the instances reflects the graph name as well as the vehicle identification number. The tables first provide the number of nodes, edges, and required edges for each of the instances, as well as the percent of nodes (% N) that were removed during preprocessing. The latter is only relevant for the CARP without duration constraints. Then, for each of the three algorithms, the tables provide the cost of the solution, the cost as percentage (%) above the best of the three (a "-" indicates that the algorithm was the better of the three), the number $|\mathcal{R}|$ of vehicles used in the solution (not available for PS), and the run time. For FastCARP, the table also provides the number $|\mathcal{D}|$ of districts generated by the algorithm. From these results, one can see how large these instances are as evidenced by the number of routes. It is also clear that even though FastCARP does not always obtain the best solution, it is superior when the run time is taken into account.

Comparing the two tables, we see that the change in cost when duration constraints are added vary from a small improvement, which is due to the fact that we are dealing with heuristics, to a doubling of the cost. This is also true for the number of routes in the solutions. Indeed, for a few instances, the solutions use the same number of vehicles, whereas for other instances, up to four times as many vehicles are used. This large increase in the number of vehicles for some of the instances reflects the impact of the route duration constraints. As a side effect, we see that the Base algorithm is generally significantly faster when duration constraints are present, due to the reduction in the number of required edges of each route.

4 Conclusions

We have developed a fast and simple heuristic called FastCARP for the solution of large-scale capacitated arc routing problems, with or without duration constraints. The development of this algorithm was motivated by the need to solve real-life waste collection problems arising in Denmark. The instances considered in this study are of large scale, containing up to 11,640 nodes, 12,675 edges, 8,581 required edges, and 323 vehicles. These sizes preclude the use of exact algorithms and even of metaheuristics since these tend to perform a deep exploration of the search space and are time consuming. FastCARP is based on a decomposition of the graph into districts which are iteratively optimized, merged, and split. This mechanism avoids working on the full graph at each iteration and yields substantial time savings. FastCARP was compared with an elementary algorithm called Base and to a number of Path Scanning variations. On small graphs it was better but slower than Base, but on larger graphs it was much faster and only slightly worse than Base in terms of solution quality. Our results indicate that FastCARP is highly suitable for the solution of large-scale instances such as those we have encountered in Denmark.

As is clear from the discussion found in (Kiilerich and Wøhlk, 2017), the

CARP that we have sought to solve in this paper is merely one of the problems that occur in kerbside collection of waste. Other problem variants include multi-compartment problems and problems where vehicle services are coordinated. Large amounts of work and challenges lies in the development of algorithms that can solve these even more challenging problems and eventually support decision makers in selection of preferred collection design.

Acknowledgements

This project was funded by the Danish Council for Independent Research - Social Sciences. Project 'Transportation issues related to waste management' [grant number 4182–00021] and by the Natural Sciences and Engineering Research Council of Canada [grant number 2015–06189]. This support is gratefully acknowledged. Thanks are due to the referees for their valuable comments.

References

Adamides E D, Mitropoulos P, Giannikos I, and Mitropoulos I. A multimethodological approach to the development of a regional solid waste management system. *Journal of the Operational Research Society*, 60(6): 758–770, 2009.

Ahr D and Reinelt R. The capacitated arc routing problem: Combinatorial lower bounds. In Corberán Á and Laporte G, editors, *Arc Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 2015.

Belenguer J M, Benavent E, and Irnish S. The capacitated arc routing problem: Exact algorithms. In Corberán Á and Laporte G, editors, *Arc Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 2015.

Brandão J and Eglese R W. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35 (4):1112–1126, 2008.

Butsch A, Kalcsics J, and Laporte G. Districting for arc routing. *INFORMS Journal of Computing*, 26(4):809–824, 2014.

Clark R M and Gillean J I. Solid waste collection: A case study. *Operational Research Quarterly*, 28(4):795–806, 1977.

Eglese R W. Routing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, 1994.

Frederickson G N. Approximation algorithms for some postman problems. *Journal of the Association for Computing Machinery*, 26(3):538–554, 1979.

Ghiani G, Mourão C, Pinto L, and Vigo D. Routing in waste collection applications. In Corberán Á and Laporte G, editors, *Arc Routing: Problems,* *Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.

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

Golden B L, DeArmon J S, and Baker E K. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

Hertz A, Laporte G, and Nanchen Hugo P. Improvement procedures for the undirected rural postman problem. *INFORMS Journal of Computing*, 11 (1):53–62, 1999.

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

Hierholzer C. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.

Khan A M. Solid-waste disposal with intermediate transfer stations: An application of the fixed-charge location problem. *Journal of the Operational Research Society*, 38(1):31–37, 1987.

Kiilerich L and Wøhlk S. New large-scale data instances for CARP and new variations of CARP. *INFOR*, Forthcoming, 2017.

Kim J-S and Lee D-H. An integrated approach for collection network design, capacity planning and vehicle routing in reverse logistics. *Journal of the Operational Research Society*, 66(1):76–85, 2015.

Letcher T and Vallero D. *Waste: A Handbook for Management.* Elsevier, Amsterdam, 2011.

Li L Y O and Eglese R W. An interactive algorithm for vehicle routeing for winter gritting. *Journal of the Operational Research Society*, 47(2): 217–228, 1996.

Mourão M C and Amado L. Heuristic method for a mixed capacitated arc routing problem: A refuse collection application. *European Journal of Operational Research*, 160(1):139–153, 2005.

Mourão M C, Nunes A C, and Prins C. Heuristic methods for the sectoring arc routing problem. *European Journal of Operational Research*, 196(3): 856–868, 2009.

Muyldermans L. *Routing, Districting and Location for Arc Traversal Problems.* PhD thesis, Catholic University of Leuven, Leuven, Belgium, 2003.

Muyldermans L and Pang G. On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, 206(1):93–103, 2010.

Muyldermans L and Pang G. Variants of the capacitated arc routing problem. In Corberán Á and Laporte G, editors, *Arc Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 2015.

Muyldermans L, Cattrysse D, and Van Oudheusden D. District design for arc-routing applications. *Journal of the Operational Research Society*, 54 (11):1209–1221, 2003.

Polace M K, Doerner K F, Hartl R F, and Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.

Prins C. The Capacitated arc routing problem: Heuristics. In Corberán Á and Laporte G, editors, *Arc Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 2015.

Prins C, Labadi N, and Reghioui M. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2): 507–536, 2009.

Usberti F L, Frana P M, and Frana A L M. GRASP with evolutionary path-relinking for the capacitated arc routing problem. *Computers & Operations Research*, 40(12):3206–3217, 2013.

Vidal T. Node, edge, arc routing and turn penalties: Multiple problems - one neighborhood extension. *Operations Research*, 65(4):992–1010, 2017.

Wøhlk S. *Contributions to Arc Routing.* PhD thesis, University of Southern Denmark, Odense, 2005.

Wøhlk S and Laporte G. Computational comparison of several algorithms for the minimum cost perfect matching problem. *Computers & Operations Research*, 87:107–113, 2017.

Zbib H. Variants of the Path Scanning Construction Heuristic for the No-Split Multi-Compartment Capacitated Arc Routing Problem. *Working Paper*, Aarhus University, 2017.