

**A fast heuristic for very large-scale
capacitated arc routing problems**

S. Wøhlk,
G. Laporte

G-2017-36

May 2017

Cette version est mise à votre disposition conformément à la politique de libre accès aux publications des organismes subventionnaires canadiens et québécois.

Avant de citer ce rapport, veuillez visiter notre site Web (<https://www.gerad.ca/fr/papers/G-2017-36>) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

This version is available to you under the open access policy of Canadian and Quebec funding agencies.

Before citing this report, please visit our website (<https://www.gerad.ca/en/papers/G-2017-36>) to update your reference data, if it has been published in a scientific journal.

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2017
– Bibliothèque et Archives Canada, 2017

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*.

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2017
– Library and Archives Canada, 2017

A fast heuristic for very large-scale capacitated arc routing problems

Sanne Wøhlk^a

Gilbert Laporte^b

^a *CORAL - Cluster for Operations Research and Logistics, Department of Economics and Business Economics, Aarhus University, Aarhus, Denmark*

^b *GERAD and Canada Research Chair in Distribution Management, HEC Montréal, Montréal (Québec) Canada*

sanw@econ.au.dk

gilbert.laporte@cirrelt.ca

May 2017

Les Cahiers du GERAD

G-2017-36

Copyright © 2017 GERAD

Abstract: The purpose of this paper is to develop a fast heuristic called FASTCARP for the solution of very 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 constructs 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

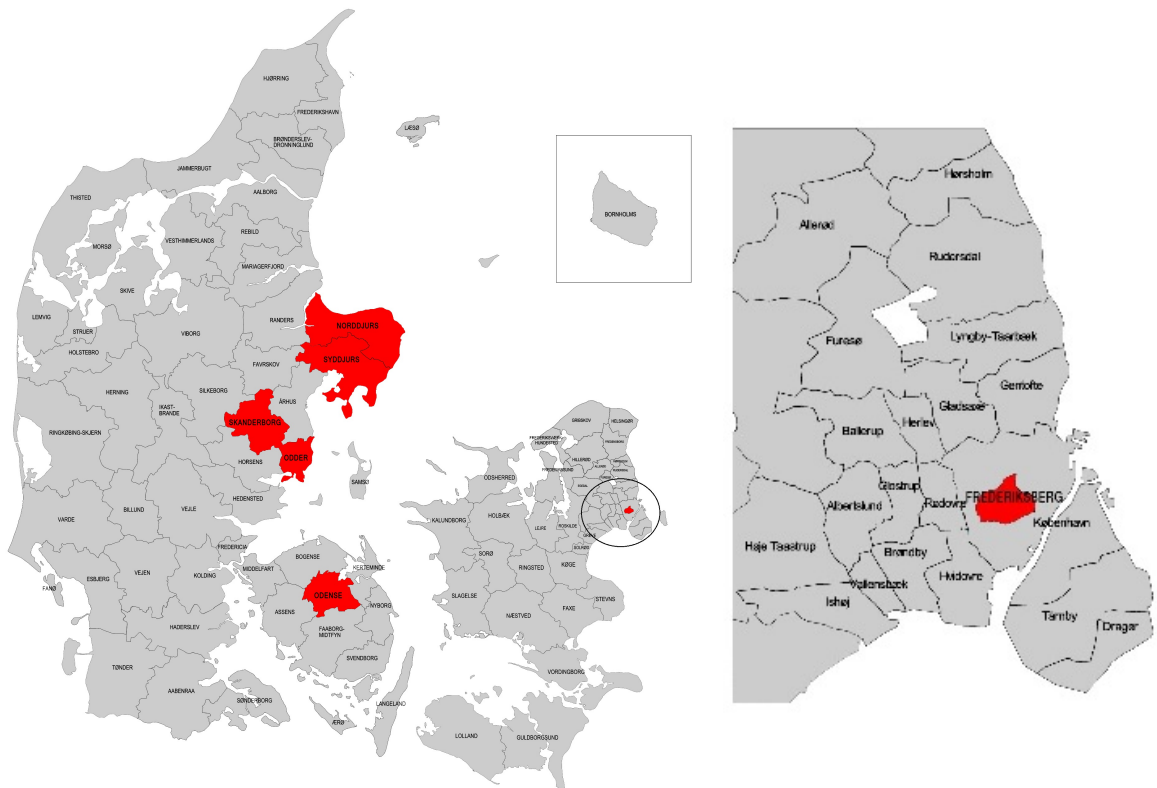
Acknowledgments: This project was funded by the Danish Council for Independent Research - Social Sciences. Project 'Transportation issues related to waste management' [grant number 4182-00021] and by the Natural Sciences and Engineering Research Council of Canada [grant number 2015-06189]. This support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

1 Introduction

The purpose of this paper is to develop a fast constructive heuristic called FASTCARP for a very 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 required edge is serviced by one of the vehicles, 2) the vehicle capacities are not exceeded by the sum of the demands q_e on their route, and 3) the total routing cost is minimised. 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.

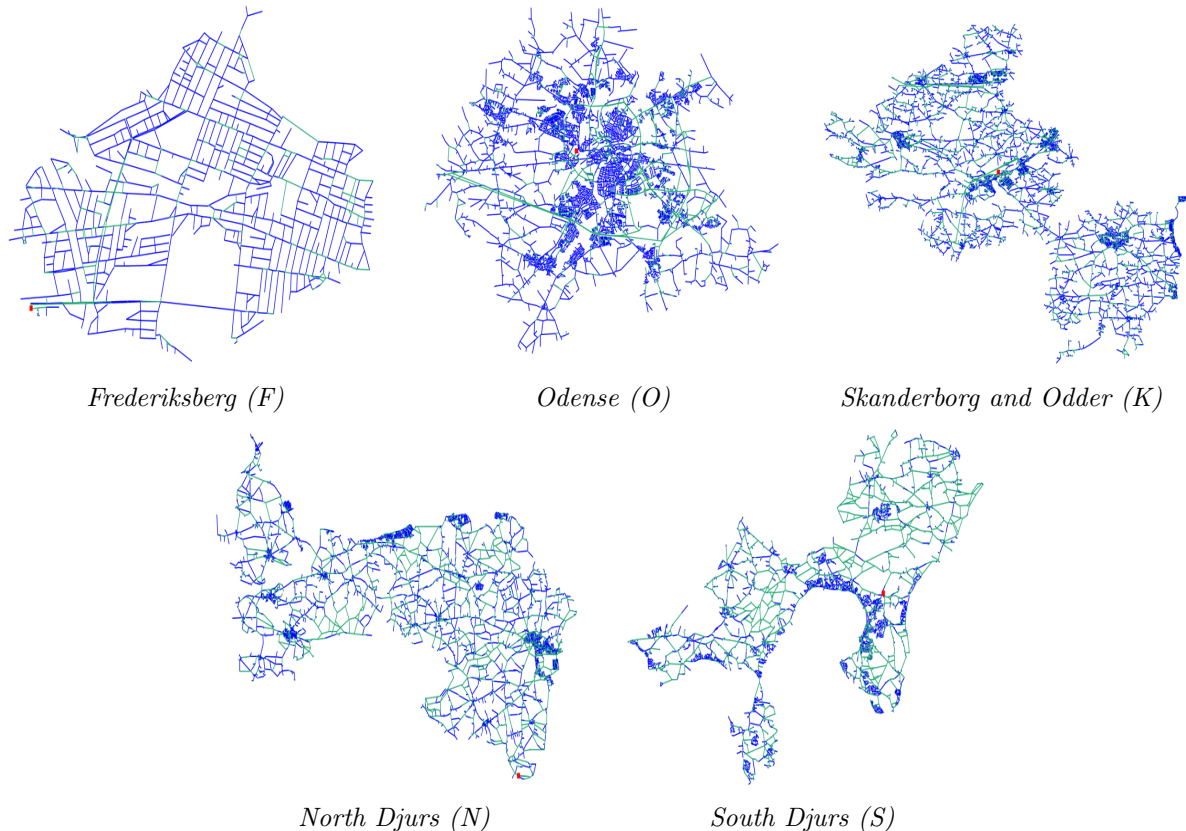


Figure 2: The five data areas. Blue edges are required and green edges are non-required. The depot is marked by a red square.

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 km² (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 order to make our contribution as generic as possible, we have made some simplifications to the problems faced by the six Danish counties. 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 very 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). See Prins (2015) for an overview of constructive heuristics for the CARP.

Given the fact that we are interested in the solution of very 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 very 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.

1.3 Contribution and organization of the paper

The contribution of this paper is to present a fast heuristic for solving very 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 very 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, which are stored in an ordered list $\mathcal{D} = (D_0, D_1, \dots, D_{|\mathcal{D}|-1})$. 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( $D$ )
   $k_1 \leftarrow k_1 + 1(\bmod |\mathcal{D}|), k_2 \leftarrow k_2 + 1(\bmod |\mathcal{D}|)$ 
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 $\mathcal{D} = (D_0, D_1, \dots, D_{|\mathcal{D}|-1})$. 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 $\beta = \lceil \sqrt{K} \rceil$ was the approach that performed best overall and this is the one that we used in the final implementation of the algorithm.

2.3 Optimisation of the districts

At each iteration of the main loop of `OPTIMISEDISTRICT`, which is repeated ψ times, 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.

Algorithm 2 OPTIMISE DISTRICT

```

procedure OPTIMISEDISTRICT( $D = (R_1, R_2, \dots)$ )
  for  $i = 1$  to  $\psi$  do
     $R \leftarrow$  PASTE( $R_1, R_2, \dots$ )
     $R, U \leftarrow$  SWITCH( $R, U$ )
     $R \leftarrow$  SHORTEN( $R$ )
     $(R_1, R_2, \dots) \leftarrow$  SPLIT( $R$ )
     $R_k \leftarrow$  SHORTEN( $R_k$ )  $\forall k$ 
  end for
end procedure

```

An essential part of `OPTIMISEDISTRICT` is the `SPLIT` procedure. 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 . 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.

3 Computational results

We have tested our algorithm on the 264 CARP benchmark instances of Küllerich and Wøhlk (2017) which consist of 88 graphs, each combined with three different vehicle capacities. The largest instance has 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 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.

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 must be 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.

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 3 depicts the graph and the edges serviced by each of the four vehicles in the solution produced by FASTCARP, and Figure 4 shows the four routes. 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 864 are required. In this solution, it can be seen that the red and the gold routes are intertwined.

Due to the very high 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. Figure 6 shows the cost obtained by FASTCARP and by the BASE algorithm as a percent above the best of these two algorithms over all instances solved as a standard CARP (left) and as a CARP with duration constraints (right). 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.

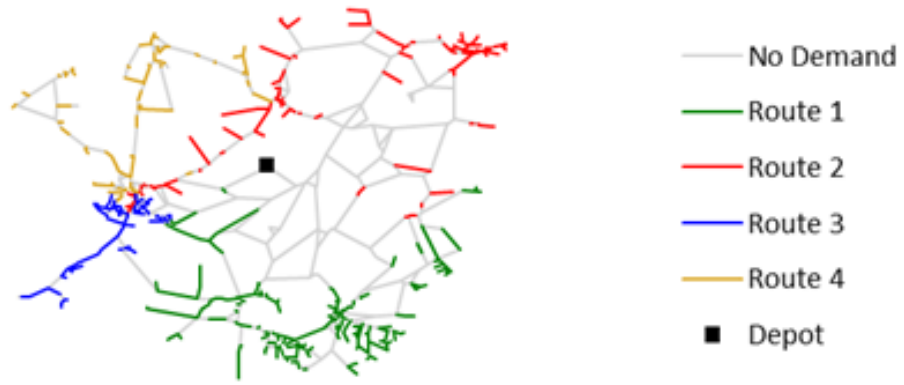


Figure 3: Example of solution. Instance S12_g-3.

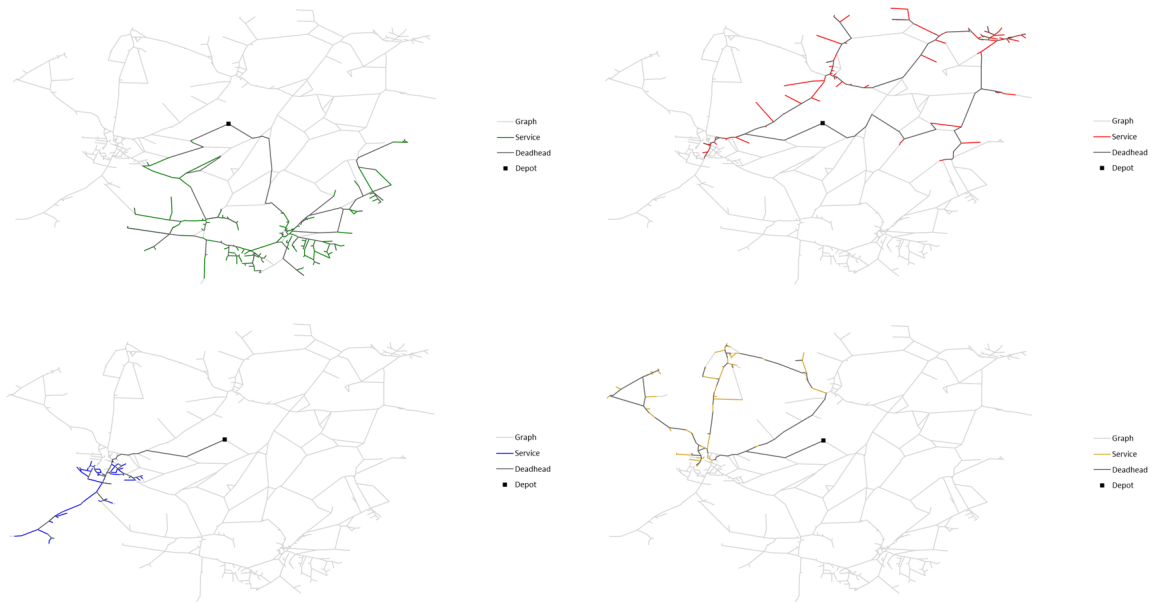


Figure 4: The four routes of instance S12_g-3 solved by FastCARP.



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

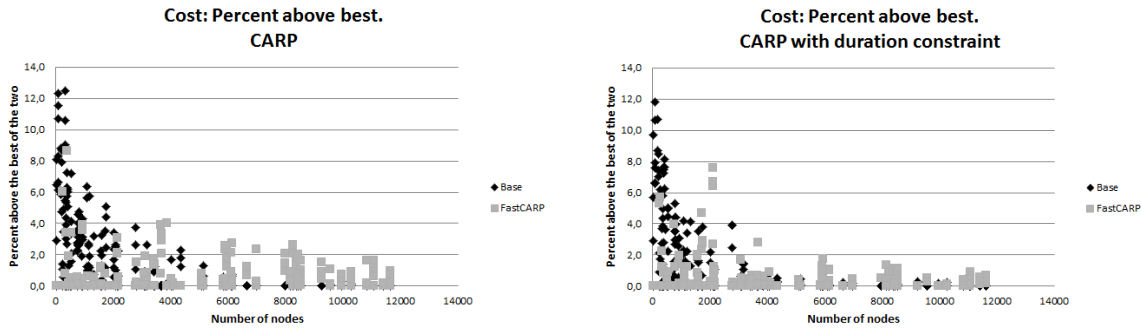


Figure 6: Cost of solutions obtained by FastCARP and the base algorithm for CARP and CARP with duration constraint.

Table 1: Summary of the computational results over all 264 instances.

	CARP		CARP with duration constraint	
	BASE	FASTCARP	BASE	FASTCARP
Average % above best	1.6	0.6	1.3	0.4
Max % above best	12.5	8.7	11.8	7.6
Total number	264	264	264	264
Number of best	125	142	133	135
Average number of routes	34.5	33.6	51.9	51.1
Max number of routes	185	185	323	230

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.

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.

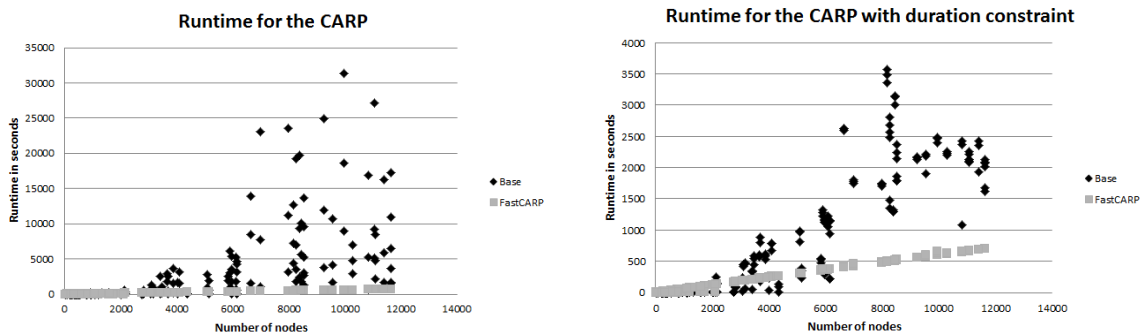


Figure 7: Run time for FastCARP and the base algorithm for CARP and CARP with duration constraint.

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 two algorithms, the tables provide the

Table 2: Computational results for CARP.

Instance	Instance information				BASE				FASTCARP				
	$ N $	$ E $	$ E_R $	$\% N$	Cost	$\%$	$ \mathcal{R} $	Time	Cost	$\%$	$ \mathcal{D} $	$ \mathcal{R} $	Time
F1_g-4	812	1124	780	1.0	804877	4.8	118	0.2	768209	-	10	113	48.7
F1_g-6	812	1124	780	1.0	491445	3.5	65	0.3	474809	-	7	63	48.8
F1_p-2	795	1106	728	1.1	272287	4.0	27	1.3	261743	-	5	27	47.7
F1_p-6	795	1106	728	1.1	163009	0.4	9	9.3	162376	-	2	8	48.0
F9_g-4	788	1096	686	1.3	751015	4.6	109	0.2	718145	-	9	105	47.3
F9_g-6	788	1096	686	1.3	458289	3.1	60	0.4	444369	-	7	59	47.3
K1_g-2	11640	12675	8566	9.3	6453427	-	150	1564.2	6501210	0.7	11	147	700.2
K1_g-6	11640	12675	8566	9.3	3711945	-	44	17254.9	3739724	0.7	6	42	698.5
K2_g-2	11636	12671	8563	9.3	6248942	-	146	1730.6	6249733	0.0	11	143	699.5
K2_g-4	11636	12671	8563	9.3	4401690	-	74	6471.6	4434203	0.7	8	71	701.0
K5_g-2	11405	12435	8267	9.3	6029327	-	140	1678.7	6031579	0.0	11	139	685.6
K5_g-6	11405	12435	8267	9.3	3571768	-	42	16328.5	3578627	0.2	6	39	685.5
N1_g-2	8537	9725	6339	7.3	6472575	-	90	1262.6	6476448	0.1	9	89	513.5
N1_g-6	8537	9725	6339	7.3	3180997	-	26	13692.1	3207125	0.8	5	26	514.0
N2_g-3	8537	9725	6339	7.3	4716390	-	57	3029.4	4749044	0.7	7	57	512.5
N2_g-5	8537	9725	6339	7.3	3436348	-	31	9566.8	3492355	1.6	5	31	516.5
N5_g-2	8444	9631	6165	7.3	5914240	0.1	82	1854.0	5908863	-	8	80	508.1
N5_g-5	8444	9631	6165	7.3	3325667	-	30	10063.5	3392187	2.0	5	29	509.7
O1_g-4	10283	11863	8581	2.8	3282866	0.1	133	2896.3	3278666	-	11	133	619.1
O1_g-6	10283	11863	8581	2.8	2700609	-	78	7035.4	2724848	0.9	8	77	618.4
O1_p-2	9957	11492	8220	3.1	2499283	-	70	8944.0	2509047	0.4	8	68	597.9
O1_p-4	9957	11492	8220	3.1	2179060	-	36	31417	2194629	0.7	5	33	600.7
O6_g-2	9563	11073	7831	2.6	3534430	0.1	158	1670.5	3531246	-	11	156	574.7
O6_g-6	9563	11073	7831	2.6	2273493	-	46	10656.6	2276829	0.1	6	43	575.4
S1_g-1	6149	7110	3797	3.7	3651318	0.7	138	78.1	3624502	-	11	137	369.3
S1_g-6	6149	7110	3797	3.7	1470679	-	21	4212.5	1478193	0.5	4	19	370.4
S2_g-1	6149	7110	3797	3.7	3480732	0.6	128	85.7	3459811	-	10	127	369.3
S2_g-6	6149	7110	3797	3.7	1439140	-	19	4685.6	1478328	2.7	4	19	369.2
S5_g-2	6104	7063	3732	3.8	2191441	-	60	684.5	2197632	0.3	7	59	367.0
S5_g-6	6104	7063	3732	3.8	1413134	-	18	5238	1443084	2.1	4	17	367.3

cost of the solution, the cost as percentage above the best of the two (a “-” indicates that the algorithm was the better of the two), the number $|\mathcal{R}|$ of vehicles used in the solution, 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 very 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 very 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

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

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

compared with an elementary algorithm called BASE. 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 very large-scale instances such as those we have encountered in Denmark.

References

- ED Adamides, P Mitropoulos, I Giannikos, and I Mitropoulos. A multi-methodological approach to the development of a regional solid waste management system. *Journal of the Operational Research Society*, 60(6):758–770, 2009.
- D Ahr and G Reinelt. The capacitated arc routing problem: Combinatorial lower bounds. In Á Corberán and G Laporte, editors, *Arc Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.
- JM Belenguer, E Benavent, and S Irnish. The capacitated arc routing problem: Exact algorithms. In Á Corberán and G Laporte, editors, *Arc Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.
- J Brandão and RW Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35(4):1112–1126, 2008.
- A Butsch, J Kalcsics, and G Laporte. Districting for arc routing. *INFORMS Journal of Computing*, 26(4):809–824, 2014.
- RM Clark and JI Gillelan. Solid waste collection: A case study. *Operational Research Quarterly*, 28(4):795–806, 1977.
- RW Eglese. Routing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, 1994.
- GN Frederickson. Approximation algorithms for some postman problems. *Journal of the Association for Computing Machinery*, 26:538–554, 1979.

- G Ghiani, C Mourão, L Pinto, and D Vigo. Routing in waste collection applications. In Á Corberán and G Laporte, editors, *Arc Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.
- BL Golden and RT Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.
- BL Golden, JS DeArmon, and EK Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.
- A Hertz, G Laporte, and P Nanchen Hugo. Improvement procedures for the undirected rural postman problem. *INFORMS Journal of Computing*, 11(1):53–62, 1999.
- A Hertz, G Laporte, and M Mittaz. A tabu search heuristic for the capacited arc routing problem. *Operations Research*, 48:129–135, 2000.
- C Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.
- AM Khan. 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.
- L Kiilerich and S Wøhlk. New large-scale data instances for CARP and variations of CARP. *INFOR*, Forthcoming, 2017.
- J-S Kim and D-H Lee. 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.
- T Letcher and D Vallero. *Waste: A Handbook for Management*. Elsevier, Amsterdam, 2011.
- LYO Li and RW Eglese. An interactive algorithm for vehicle routeing for winter gritting. *Journal of the Operational Research Society*, 47(2):217–228, 1996.
- MC Mourão and L Amado. Heuristic method for a mixed capacitated arc routing problem: A refuse collection application. *European Journal of Operational Research*, 160(1):139–153, 2005.
- MC Mourão, AC Nunes, and C Prins. Heuristic methods for the sectoring arc routing problem. *European Journal of Operational Research*, 196(3):856–868, 2009.
- L Muyldermans. *Routing, Districting and Location for Arc Traversal Problems*. PhD thesis, Catholic University of Leuven, Leuven, Belgium, 2003.
- L Muyldermans and G Pang. On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, 206(1):93–103, 2010.
- L Muyldermans and G Pang. Variants of the capacitated arc routing problem. In Á Corberán and G Laporte, editors, *Arc Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.
- L Muyldermans, D Cattrysse, and D Van Oudheusden. District design for arc-routing applications. *Journal of the Operational Research Society*, 54(11):1209–1221, 2003.
- M Polacek, KF Doerner, RF Hartl, and V Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.
- C Prins. The Capacitated arc routing problem: Heuristics. In Á Corberán and G Laporte, editors, *Arc Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.
- C Prins, N Labadi, and M Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47:507–536, 2009.
- FL Usberti, PM Frana, and ALM Frana. GRASP with evolutionary path-relinking for the capacitated arc routing problem. *Computers & Operations Research*, 40(12):3206–3217, 2013.
- T Vidal. Node, edge, arc routing and turn penalties: Multiple problems - one neighborhood extension. *Operations Research*, Forthcoming, 2017.
- S Wøhlk. *Contributions to Arc Routing*. PhD thesis, University of Southern Denmark, Odense, 2005.
- S Wøhlk and G Laporte. Computational comparison of several algorithms for the minimum cost perfect matching problem. Submitted for publication, 2017.