Winter Gritting Routes with Multiple Visits

A heuristic for a capacitated arc routing problem with heterogenous vehicles with different covering widths.

Master′s thesis

Lasse Damtoft Nielsen

Aalborg University
Mathematics-Economics

AALBORG UNIVERSITY
STUDENT REPORT

Mathematics-Economics
Aalborg University
http://www.aau.dk

Title:
Winter Gritting Routes with Multiple Visits

Subtitle:
A heuristic for a capacitated arc routing problem with heterogenous vehicles with different covering widths.

Project Period:
Fall Semester 2019

Student:
Lasse Damtoft Nielsen

Supervisor:
Peter Nielsen
Inkyung Sung

Page Count: 28

Date of Completion:
December 19, 2019

Abstract:

In this project we address the practical aspects of solving the problem of winter gritting, as a Capacitated Arc Routing Problem (CARP). Specifically, based on discussions with COWI, and a literature review we find two main gaps in the current literature, and focus on the most important of these. That is the fact that the number of times a vehicle needs to visit a road to service it depends on the specific vehicle. We then proceed to propose a solution heuristic which can handle this type of problem, i.e. determining which roads should be serviced by which vehicles. The proposed solution utilizes existing methods for the sub-problems that are well researched, while introducing a method for the multiple visits. The parameters of the proposed solution are tuned and we test our heuristic on test instances from literature. We find that the proposed solution works and while there are no existing solutions in the literature to compare our results to, we find potential for improvement which we address at the end.

Contents

Preface

This project has been carried out in the final semester of the Mathematics-Economics Masters degree with specialization in operational research.
I want to thank my supervisors, Peter Nielsen and Inkyung Sung, for their devoted help on this project. I would also like to thank COWI for the cooperation, especially, a thanks to Ulla L. Sørensen and René Carlsbæk Gundersen for their help through the project.

Aalborg University, December 19, 2019

Lasse Damtoft Nielsen
<ldni14@student.aau.dk>

Chapter 1

Introduction

To set the scene of this project lets go back in time to the city of Königsberg (now Kaliningrad) which in the 18th century had seven bridges, joining four landmasses as shown in Figure 1.1. The people of Königsberg wanted to find a route that would cross each bridge exactly one time. This problem was formalised by Leonhard Euler in [1], which laid the foundation for graph theory. Euler proved that the problem could not be solved as each landmass had an odd number of bridges. His work in this field inspired the terms Eulerian path and Eulerian cycle or circuit, for when it is possible to traverse each arc of a graph exactly one time. Eulerian path for when it is not possible to start and end in the same node and cycle or circuit when it is possible.

Many years later, in 1962 this problem was further elaborated into the Chinese Postman Problem (CPP) by Mei-Ko Kwan [2]. The problem got its name from the problem faced by postmen who needed to cover all roads in a given region while travelling the minimum total distance. Similarly to the Könisberg bridge problem, the problem is to cover all arcs in a graph, however, accepting that some must be traversed more than once and this extra distance is sought minimized. In 1974 C.S. Orloff further suggested the Rural Postman Problem (RPP) where only

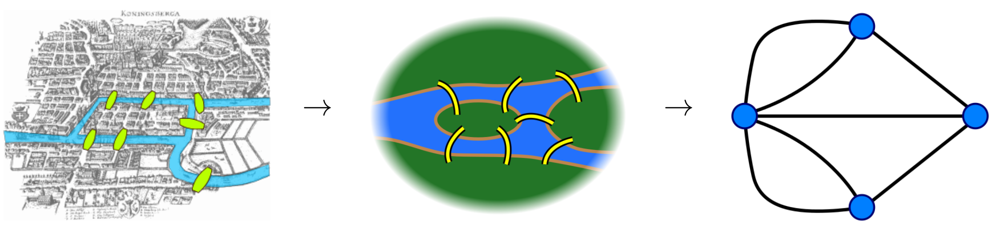

Figure 1.1: The bridges of Königsberg

a subset of the arcs needs to be visited [3]. Since then a number of variations has been suggested including the Windy Postman Problem (WPP) where the cost of arcs are different depending on the direction it is traversed in. Finally, we arrive at the focus of this project namely the Capacitated Arc Routing Problem (CARP), which was introduced in 1981 by Bruce L. Golden and Richard T. Wong, which is the larger problem of the RPP where multiple routes should be constructed (in the postman setting representing multiple postmen sharing the job) and include a capacity for each route (in the postman setting a maximum load of mail each postman can carry on the route) [4].

1.1 Problem description

Today the CARP remains relevant to describe problems faced in many aspects of both public and private businesses, where street segments rather than specific points need service, e.g. waste collection, street sweeping, snow ploughing, or as is the case in this project salt spreading, which is also known as the Winter Gritting problem. In the winter of countries where snow is common, a problem faced by municipalities is that of winter gritting. That is the need to spread salt or sand on the roads to ensure traffic safety. This problem is for example faced by many municipalities in Denmark to construct routes for contractors who then does the winter gritting using a number of gritting vehicles starting from a central depot. The winter gritting problem is typically formulated as a CARP. The objective is to minimize the total routing cost or time consumption and the CARP has been proven to be $\mathcal{NP}$-hard [5].

One company that is faced with this problem is COWI, which "is a leading consulting group" [6] who does this planning for some municipalities in Denmark. In the beginning of this project we had a meeting with COWI where we gained some insights into the practical aspects of winter gritting they were facing. Even though winter gritting is a fairly well researched problem, we believe the real-life problem has some aspects that has not been researched thoroughly. The situation that must be addressed by COWI is highly complex. It is primarily constrained on the following factors. The problem is that of winter gritting a set of roads in a municipality. The roads are sorted into several categories as for the type of road, as well as the priority of having it gritted. For simplicity we will stick with one naming convention namely categories, but it should be noted that there is an implied prioritization in this term. This should be done with a heterogeneous set of vehicles, where some vehicles may need to service a given road more than once, depending on the particular equipment (width with which it can spread salt) and the width of the road. Furthermore, some vehicles may need to only travel the roads they have been classified for servicing (for example, tractors should never use the highway), while others may use other category roads. This is called the vehicle road segment dependency. This last part is both due to the nature of each vehicle but also the number of complaints the municipality receives from citizens when they see a gritting vehicle traversing a road that is not yet serviced (or being serviced at all). Finally the time it takes to deadhead (traverse but not service) a road may change when it has already been serviced. However one thing that it does not need to handle is directed arcs, as these are not commonly seen in the same areas where the rest of these problems occur. We can therefore focus only on the setting where roads can be traversed and serviced in both directions. To sum up, we think that a practical real-life solution to the winter gritting problem should be able to handle:

- Different categories of roads.
- Differences in road width and spreading width of the vehicles, and thus the number of times a vehicle needs to traverse a road in order for it to be serviced.
- Deadheading time change when serviced.
- Heterogeneous fleet including the vehicle road segment dependency.

A practical real-life solution to the winter gritting problem should be able to handle these different additions to the CARP, as well as being reasonable in computational speed. The reason that computational speed is a factor is both that a set of roads which needs service, may be subject to day-to-day change, as the whether changes, and that the results should be available within a reasonable time in order to "prototype" long-term changes, for example moving the depot, expanding or decreasing the fleet or up- or downgrading the equipment on vehicles or changing the category of certain roads. We believe that these additions to the CARP has not been researched fully and we will now briefly introduce some of the literature in order to back up this belief.

1.2 Literature Review

Several methods dealing with the CARP already exist. In [5] and [7] many of these are presented and discussed. In this section I will limit my review to those methods designed for a variation of the winter gritting problem similar to the problem we seek to research.

The problem of having several categories of roads is well known and a number of ways to handle this has been proposed [8]. One common way of handling this problem is called linear precedence relation, which means that the most important roads must be serviced before any of the next category and so on. This is a hard constraint and the slightly weaker version where the high priority must be serviced before any of the low priority, but where medium can be done at any time is called the general precedence relation, and several algorithms has been proposed in the case of each type of constraint. A factor which plays an important role in the ability to solve these types of problems efficiently is that of connectivity. When splitting the network of roads into sub-graphs there is no guarantee that these sub-graphs are connected (while we expect the whole network to be connected). If all the sub-graphs induced by the categories are connected there exist polynomial time solutions, however this is not generally the case in real-life [8]. As this type of problem is well researched, and the winter gritting problem has a linear precedence relation, we will disregard this part of the problem in our solution and state that it should be made in a way that allows it to solve the problem on each of the distinct categories of roads, while allowing for some deadheading on other categories in order to ensure connectedness of the sub-graphs induced by each category.

The aspect that deadheading an arc may not take a fixed amount of time, but may vary if the arc has been serviced beforehand (it should be faster to deadhead it later on), is presented in [9], where they also present a solution heuristic that is able to handle this variation. They also include having different cost for different travel directions to account for ploughing uphill or downhill, and even though our problem is not that of ploughing there may be some cases where this property applies to salt trucks as well. However [9] only considers a single vehicle does not include having different categories of the given roads, as well as not having a limited capacity on the vehicle.

Thirdly the problem of the heterogeneous fleet is described in [10] that also considers having multiple depots, which could also be the case in the problem in real-life. The vehicles would usually be largely decentralized as they are often owned by private contractors who then do the servicing for the municipality, however this can be disregarded as the vehicles would need to start at the depot in order to be loaded with salt. While they fail to consider different categories of roads this could be fixed by solving the different priority classes separately, which is equivalent to solving the problem using linear precedence relation and assigning the vehicles to the highest priority class they can service. However [10] (as all of the above) consider the cost of the road to be fixed as a mapping of the length, and this is not the case if the heterogeneous fleet of vehicles travels at different speeds (tractors being slower than trucks) which means that the cost of a road in practice can not be considered constant. In [11] they introduce different costs for different types of trucks, as well as a matrix to indicate whether some types of trucks can service other classes than their own. This is very similar to the problem that we address (for the heterogeneous fleet part) however this model is made with waste collection in mind and it thus do not include changes in deadheading time or categories of roads.

Furthermore the heterogeneity of the vehicles in practical cases also introduces the need for some vehicles to do several passes of a road, in order to consider it serviced, where other vehicles may be able to service it in one pass. This is touched on in [12] for Arc Routing Problems (ARP) where they include two types of vehicles, where one may need to service an arc twice, however as it is for ARP they do not include capacities for the vehicles and only include two types of vehicle for snow grooming. In our case, the capacity is an important part of the problem and we cannot limit it to a pre fixed number of vehicle types. Thus this is not a type of problem that we have been able to find similar in the literature.

With this review, which is summed up in Table 1.1, we have identified some of the remaining problems in solving the winter gritting problem. While some of the problems has been addressed in the literature there do not exist a single solution for this type of problem. We therefore seek to combine what has already been addressed in a new way, to fit this problem, along with solving some of the remaining problems, that has not yet been addressed. While both the multiple visits and the change in deadheading speed parts of the problem deserve to be researched further, doing both would be outside the scope of this project. From our meeting with COWI, we found out that the speed change does not impact their routes as often and therefore, the most important of the two is allowing for multiple visits. We thus limit the scope of this project to solving the problem of having multiple visits to a given road segment, if the allocated vehicle cannot service it in a single visit.

Reference	Categories	Multiple Visits	Speed Change	Heterogeneous Fleet	Capacity
[9]	No	No	Yes	No	No
[10]	No	No	No	Yes	Yes
[11]	No	No	No	Yes	Yes
[12]	Yes	Two	No	Yes	No
COWI's problem	Yes	Any number	Yes	Yes	Yes
Proposed solution	Yes	Any number	No	Yes	Yes

Table 1.1: Overview of reviewed references

The remainder of this project will be structured as follows: In Chapter 2 we will give a more detailed and mathematical description of the introduced problem. In Chapter 3 we will propose a solution algorithm able to handle the CARP with potential multiple visits. In Chapter 4 we tune the algorithm as well as analyse results from using it. Finally we will conclude on our findings in Chapter 5. Please note that we assume that the reader is familiar with basic graph theory and the terminology of this.

Chapter 2

Modelling

In order to solve the problem addressed in this project, we would first need to have a proper model for it, so that we are clear what the constraints are and what needs to be optimised.

Let the graph $G = (V, A)$ represent the road network, where V is the vertices (traffic junctions) and A is the arcs (roads), now let $A_R \subseteq A$ represent the arcs in G that require service. Now let $p \in P$ be the possible arc sets and $k \in K$ be the vehicles. Then each set of arcs p can be allocated to a vehicle k from which we can calculate the cost C_p^k of servicing route p with vehicle k. This C_p^k is the time it takes to service all the arcs in set p with vehicle k including any necessary deadheading time, however not including the time to return to the depot, this cost can be found by solving a RPP. Let Z_p^k be the binary decision variable, that is 1 when arc set p is allocated to vehicle k, and 0 otherwise:

$$Z_p^k = \begin{cases} 1, & \text{if arc set } p \in P \text{ is allocated to vehicle } k \in K \\ 0, & \text{if arc set } p \in P \text{ is not allocated to vehicle } k \in K. \end{cases} \tag{2.1}$$

This allows us to do a set-partition formulation of the problem as:

$$\min Z \tag{2.2}$$

s.t.

$$Z \geq \sum_p C_p^k Z_p^k \quad \forall k \tag{2.3}$$

$$\sum_K \sum_P a_{ij} Z_p^k = 1 \quad \forall (i, j) \in A_R \tag{2.4}$$

$$d_p Z_p^k \leq Q_k \quad \forall k \in K \tag{2.5}$$

$$\sum_P Z_p^k \leq 1 \quad \forall k \in K \tag{2.6}$$

$$Z_p^k \in \{0,1\} \quad \forall k \in K \quad \forall p \in P \tag{2.7}$$

Where d_p is the total demand of arc set p, Q_k is the capacity of vehicle k, and a_{ij} an indicator variable that there is an arc connecting nodes i and j. First we have the objective function in (2.2), which is directly constrained by (2.3). This formulation states that we seek to minimize the time it takes for the last vehicle to finish its job. This objective is chosen because we need all the required arcs to be serviced, however we do not gain any additional value from being faster back at the depot and thus there is no reason to include this travel time in the minimization. This objective is also what is used by COWI, where their contract is to make sure that the roads are gritted in a certain time-frame, but does not include that the vehicles have to return to the depot in this time. A Figure of the chosen objective is shown in Figure 2.1. We refer to this as makespan optimization. In (2.4) we make sure that a solution needs to service all roads in A_R. In (2.5) we ensure that no route needs more salt than the capacity of the given vehicle. In (2.6) we ensure that no more than one set of arcs are allocated to a single vehicle, and (2.7) is just that Z_p^k is a binary variable.

We now have a formal description of the problem we seek to solve and are ready to propose a solution method.

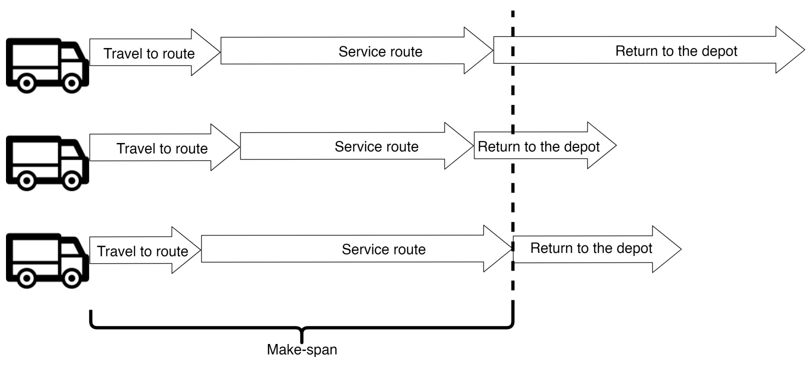

Figure 2.1: Decomposition of vehicle routes, our objective is the dotted line

Chapter 3

Proposed solution

When looking to solve a problem as the one proposed in this project which is $\mathcal{NP}$-hard we typically look towards algorithms. Three types of algorithmic methods dominate the field, they are, exact solutions, meta-heuristics and classical heuristics. Exact solutions however only work for small problem instances of the problem and thus we disregard this type, as we seek a solution that works for large-scale problems. Meta-heuristics most often require a large number of iterations to obtain high quality solutions and are thus time consuming for large problem instances [13]. However we cannot say whether the type of solution we are aiming for in this project is reachable with metaheuristics as the computational speed vs quality of solution trade-off, in a decision support system has less emphasis on the speed.

The way (meta)heuristics works is that we start with some solution which is evaluated and updated, usually with some predefined way of ensuring convergence but sometimes also with some randomness to allow for exploration of the solution space. This is repeated until some pre-defined convergence criterion is reached or a time limit is met. Our goal is to reach the best allocation of arcs to specific vehicles as well as the routes going through the graph that covers these arcs in the least amount of time. We will do this with a genetics inspired algorithm.

The way we propose to solve this problem is to start with some (initially random) arc allocation, i.e. a set of arcs is allocated to each vehicle in accordance with their compatibility. Then from this allocation a route for each vehicle is found, which is then evaluated with our objective function (2.2). Then the arc allocation is updated and the process is repeated. The way we update our solution is to perform a neighbourhood search and at each step a population of the best solutions are kept and used in the next iteration, this process is repeated until there are no improvement is found. A flowchart of this procedure is shown in Figure 3.1. Because of the implied prioritization in the classification of roads, higher classification roads

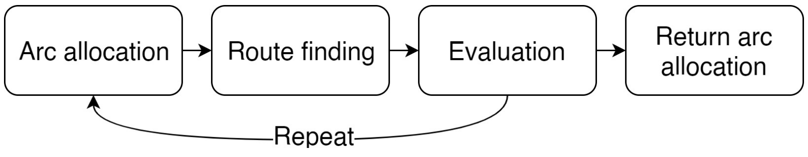

Figure 3.1: Flowchart of proposed solution

will be serviced with all available vehicles before allocating vehicles to lower classification roads. This means that the algorithm should simply run isolated on the distinct categories with the vehicles available, and thus there is no algorithmic need to incorporate categories into the solution. However note that one may need to add arcs to the graphs induced by the different categories to ensure connectedness. This is how it is handled in much of the literature, and COWI confirms this method, as higher categories of roads are much more important. We will now go through the steps of the algorithm to further give an understanding of the procedure, a pseudo-code of the algorithm is included in Algorithm 1, note that this is the whole proposed solution and that which part of the pseudo-code belongs to which parts of the solution, is shown in the caption.

Arc allocation

Arc allocation is the procedure of assigning each arc to a vehicle, which is in terms of our optimization problem to determine the which Z_p^k should be 1. This can be done in a number of ways, but for an initial population we will simply do it randomly, while making sure that the capacity constraint of each vehicle is satisfied, as well as the vehicle road segment dependency. The way we update in this algorithm is a neighbourhood search, specifically for each solution in the population we try assigning each arc to each of the other vehicles available. This forms a rather large tree which is then pruned based on a number of checks that are much computationally cheaper than to find the routes. First of course is whether it is feasible, so if it keeps the capacity constraint and is not using vehicles that are not classified for specific arcs. Second is the fixed cost, as it is possible at the point of arc allocation to know some of the value of the objective function. Namely how much time is spent servicing these arcs plus the time spent on connecting them, but without counting the time to get to the route and the deadheading encountered on the route. If the fixed cost is already higher than the best known solution then we do not need to determine the routes of this allocation to know that it is inferior. Third we have included an index to grasp the dispersion of the graph, i.e. how connected it is and thus serve as a measure of how efficiently it makes use of resources. This is the number of disconnected sub-graphs in an arc allocation, divided by the number of arcs that needs to be served. This is the reciprocal of the number of arcs per sub-graph, and in our implementation the limit is set to 0.5.

```
   Input: graph, vehicles, pop_size, N_mutations
1  population = random allocation $\times$ pop_size ;
2  best = best of population;
3  old = best+1;
4  while best < old do
5      old = best;
6      neighbourhood = neighbours(population);
7      for $i \leftarrow 1$ to N_mutations do
8          add random solution to neighbourhood;
9      end
10     for solution in neighbourhood do
11         if pruning_checks(solution) then
12             delete from neighbourhood;
13         else
14             keep in neighbourhood;
15         end
16     end
17     for solution in neighbourhood do
18         for vehicle in vehicles do
19             route_vehicle = solve RPP;
20             value_vehicle = evaluate route_vehicle;
21         end
22         value_solution = max(value_vehicle);
23     end
24     if any(value_solution < population) then
25         update population;
26         if any(population < best) then
27             update best;
28         end
29     end
30 end
```

Algorithm 1: Pseudocode for proposed solution. Lines 1-3 is the initialization, lines 6-16 is the arc allocation, lines 17-23 is the route finding and lines 24-29 is the evaluation.

Thus meaning that if each disconnected sub-graph on average contains less than two arcs then the solution is not efficient. It may be reasonable to set this number even lower for large problems, however as we are mainly interested in pruning we keep this at 0.5 to not cut too many potential solutions. Furthermore, a number of random potential solutions are added at each step in an attempt to explore more of the solution set.

Route finding

The process of route finding is to solve a RPP, which is in general $\mathcal{NP}$-hard [14], for each vehicle, i.e. to find the route covering the specific arcs allocated to each vehicle, with the shortest total travel-distance. To solve this RPP we use two heuristic solutions and pick the better of their two results. The first heuristic is the so called Frederickson heuristic [15] and the second heuristic is one that is inspired by Frederickson but which does not form an entire Eulerian cycle but only a path, as our objective is to minimize the time until the job is done, not including the time to return to the depot. The Frederickson is well explained in [14]. The process of the other heuristic is to first make the graph of the required arcs into a connected graph. The is done iteratively by first looking at one of the connected sub-graphs and from this find the shortest path from any node in this sub-graph to any node in any other connected sub-graph. When this is found, the shortest path is included into the required arcs and the two connected sub-graphs now form one connected sub-graph, these steps are then repeated until all of the required arcs are connected. Of course if the sub-graph from an arc allocation was already connected then this step is skipped.

Once a connected graph is acquired the next step is to make it have Eulerian paths or cycles. This is again done iteratively by looking at the nodes with odd degree and determining the shortest path between all of these, then the minimum of these shortest paths is duplicated and the two nodes it goes from and to are now even degree. This process is repeated until only two nodes have odd degree. Now we have a connected graph with no more than two nodes having odd degree and thus there exists a Eulerian path [16].

At this point we can find the Eulerian paths, however it may not be optimal, as the two endpoints are fixed since we have two nodes with odd degree, we therefore compare the cost of travelling from the depot to the nearest of these endpoints to the cost of adding the shortest path between them to the graph and then instead going to the node nearest to the depot and starting the route from there. Once this step is done we have a route for the RPP and the cost of this is compared to the outcome of the Frederickson algorithm and the better route is used. We note that if some of the road segments allocated to a given vehicle leads to the need of multiple visits, we would simply duplicate this road segment and require all duplicates of it to be serviced, and thus still have a RPP. Of course many other heuristics than the Frederickson algorithm has been developed through the years, however the main focus of this project is not to provide a solution to the RPP and thus we use the Frederickson algorithm and note that some of the many other algorithms could readily be used in its place.

Evaluation

Given now a set of routes we know the C_p^k's and can thus determine the value of the current solution from (2.2). At this step we further evaluate which part of the neighbourhood should replace part of the current population and if updates should be made to the best known solution. If no better solution is found in the neighbourhood, then the loop breaks and we return the currently best known allocation.

Note that we have two parameters in this process, namely the population size and the number of mutations.

3.1 Example

I will now go through a simple example of how the mechanics of the described heuristic works. The example graph is shown in Figure 3.2. The arcs of this graph are also represented in Table 3.1 in order to better explain the steps. Lets say we have two vehicles as described in Table 3.2. Both vehicles are classified to grit all road segments, which is of the same classification. The capacity of the vehicles are far greater than what is spent in this example so this is not included, however it is a fairly simple check to calculate the salt usage vs the capacity of the particular truck. I will start with a single arc allocation and go through route finding and evaluation for this, and then go back to arc allocation to show how it is updated.

3.1.1 Arc allocation (i)

We start with a random arc allocation, lets say that vehicle one grits arc, 1,3,6,8,9,11, and 13, and vehicle two does the rest of them. This gets us two sub-graphs that should be covered by the two vehicles. These sub-graphs are shown in red in Figure 3.3.

3.1.2 Route finding

Step two is to determine a route for each of these sub-graphs that starts in the depot. It is clear that even at this relatively simple example there is no clear route that is the best for each of the vehicles. Given this arc allocation we see that vehicle one has three disconnected sub-graphs, i.e. in terms of arcs $(1,3)$, (6), $(8,9,11,13)$,

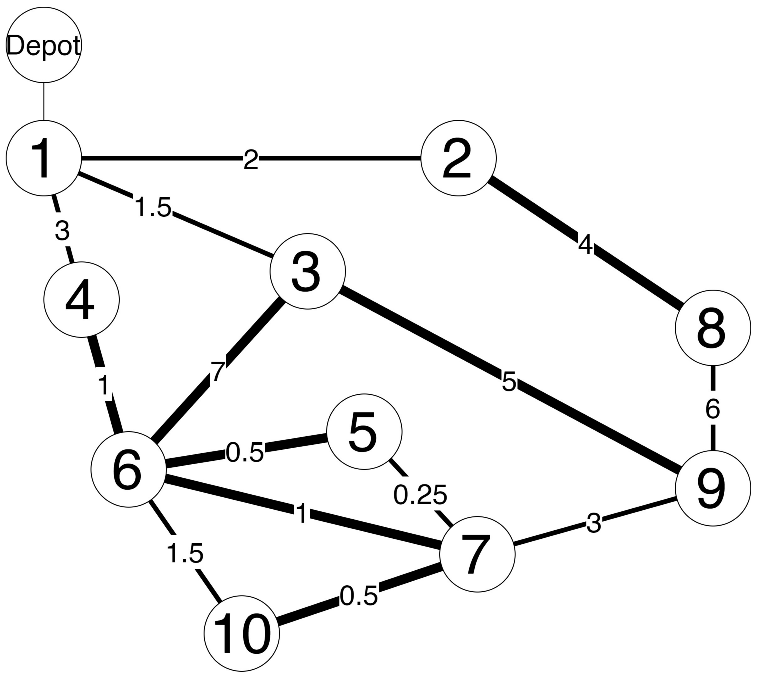

Figure 3.2: Example graph, note that the width of the arcs are different, the numbers on the arcs represent the lengths, this graph is further illustrated in Table 3.1

Arc	Nodes	Length	Width
1	1-2	2	3
2	1-3	1.5	3
3	1-4	3	3
4	2-8	4	6
5	3-6	7	6
6	3-9	5	6
7	4-6	1	6
8	5-6	0.5	6
9	5-7	0.25	3
10	6-7	1	6
11	6-10	1.5	3
12	7-9	3	3
13	7-10	0.5	6
14	8-9	6	3

Table 3.1: Table representation of arc in graph from Figure 3.2.

ID	Spreading Width	Service Speed	Deadhead Speed
1	3	15	30
2	6	30	60

Table 3.2: Vehicle list.

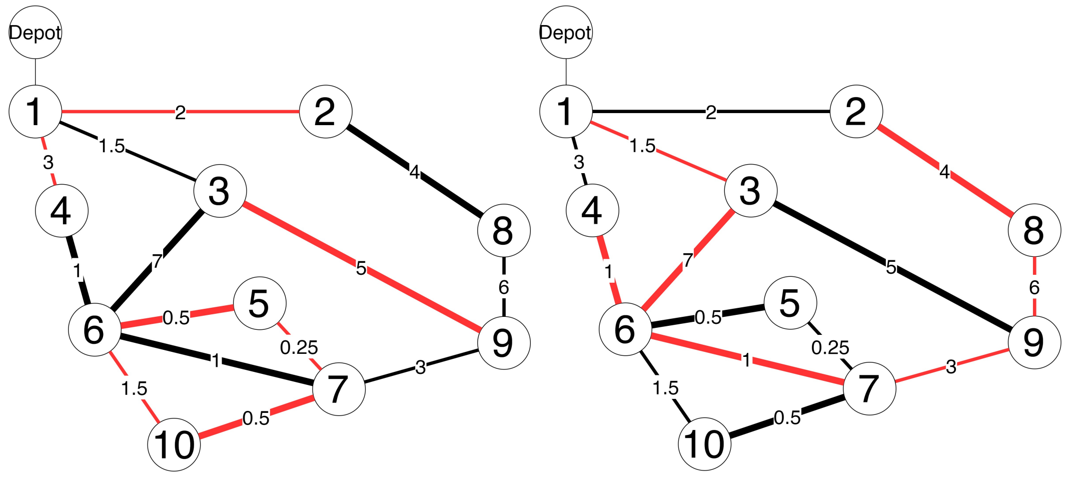

Figure 3.3: Sub-graphs formed by arc allocation

looking at the sub-graph $(1,3)$ we see that it can either be connected to (6), at a cost of 1.5 or to $(8,9,11,13)$ at a cost of 1 and thus it is connected to $(8,9,11,13)$ by adding arc number 7. Next is now to connect $(1,3,7,8,9,11,13)$ to (6) which is done cheapest at 1.5 by adding arc 2, and thus the connected sub-graph now spans $(1,2,3,7,8,9,11,13)$.

Next we need to make this sub-graph Eulerian, note that every arc with a width of 6 is included twice to ensure that is it serviced twice as vehicle one only has a width of 3. The degrees of the nodes are shown leftmost in Table 3.3. So to make the graph Eulerian we will go through the process of the heuristic that is not Fredericksons, as Fredericksons has been explained thoroughly, through the years. We start by looking at the shortest paths between each of the odd degree nodes, this is shown at the top in Table 3.4, and pick the minimum value. In this example we first add a path from 5 to 7, and in this case that is the specific arc, however it could be multiple arcs and it would not have to be from the set of arc that needs service. This gives us a new node table shown in the middle of Table 3.3 and new table of shortest paths between odd degree nodes in the bottom of Table 3.4. Here the shortest path is from 1 to 3 and thus this is added giving us another new node table where we can see that there now is exactly two nodes with odd degree and since this graph is connected, there must exist a Eulerian path.

Next we must determine whether to use the Eulerian path or to add the final

Node	Degree
1	3
2	1
3	3
4	2
5	3
6	4
7	3
9	2
10	3

$\rightarrow$

Node	Degree
1	3
2	1
3	3
4	2
5	4
6	4
7	4
9	2
10	3

$\rightarrow$

Node	Degree
1	4
2	1
3	4
4	2
5	4
6	4
7	4
9	2
10	3

Table 3.3: The degrees of nodes in the connected sub-graph for vehicle one.

Nodes	1	2	3	5	7	10
1	0	2	1.5	4.5	4.75	5.25
2	2	0	3.5	6.5	6.75	7.25
3	1.5	3.5	0	6	6.25	6.75
5	4.5	6.5	6	0	.25	.75
7	4.75	6.75	6.25	.25	0	.5
10	5.25	7.25	6.75	.75	.5	0

$\downarrow$

Nodes	1	2	3	10
1	0	2	1.5	5.25
2	2	0	3.5	7.25
3	1.5	3.5	0	6.75
10	5.25	7.25	6.75	0

Table 3.4: Shortest paths between odd degree nodes.

connection between 2 and 10 and thus having Eulerian cycles. First we determine the point in the sub-graph closest to the depot, in this case it would be 1, and check if a Eulerian path starts here, if it does then this is used as the route. If no Eulerian path starts in 1, as in this case where the endpoints are 2 and 10, then we first check the minimum distance from the depot to either of these points, in our case this is node 2 which adds a distance of 2 to the route. This is compared to the cost of connecting 2 and 10 which is 7.25 (as the connection of nodes (2-1-4-6-5-7-10), has a total length of 7.25) and since it is shorter to go to 2 and start from there, that is what we do. From this we can get the following route (1-2-1-3-9-3-1-4-6-5-7-10-7-5-6-10) which is the route for vehicle one given this arc allocation.

The arcs for vehicle two are already connected, so that step is skipped, furthermore it is fairly trivial to see that by duplicating the arc from 4 to 6 a Eulerian path is readily available starting in 1 and ending in 2 so this gives us the route for vehicle two (1-3-6-4-6-7-9-8-2).

That way all road segments are served (some are served twice by vehicle one) and every subsequent visit after servicing a road segment will be at deadheading speed.

3.1.3 Evaluation

With these routes in hand it is a simple calculation of dividing the length of each segment by the speed at which is it done to get the time consumption of each segment which is then summed for each vehicle to determine the total time of each vehicle. The routes and speed at which the road segments is carried out is shown in Tables 3.5 and 3.6. Total time of vehicle one is 87.5 minutes and for vehicle two it is 48 minutes, thus for our objective function the value of this solution is 87.5 minutes, as we seek to minimize the time spent until the last job is done. Since no previous best solution exists at this point we repeat.

3.1.4 Arc allocation (ii)

While what we have just illustrated was just for a single arc allocation this procedure should be carried out for a population and the neighbourhood of this population. To illustrate this neighbourhood lets say the arc allocation used above is the population and thus this is with a population size of one, which is a configuration one could use in the algorithm. The neighbourhood is found by allocating each arc in each of the members of the population to each of the other available vehicles. In this case with two vehicles it would simply mean moving each arc from one vehicle to the other, but with more vehicles comes alot more neighbours. In Table 3.7 the current population is shown, along with all of the neighbours.

Arc	Length [Km]	Speed [Kph]	Time [minutes]
1	2	15	8
1	2	30	4
2	1.5	30	3
6	5	15	20
6	5	15	20
2	1.5	30	3
3	3	15	12
7	1	30	2
8	0.5	15	2
9	0.25	15	1
13	0.5	15	2
13	0.5	15	2
9	0.25	30	0.5
8	0.5	15	2
11	1.5	15	6

Table 3.5: The routes of vehicle one.

Arc	Length [Km]	Speed [Kph]	Time [minutes]
2	1.5	30	3
5	7	30	14
7	1	30	2
7	1	60	1
10	1	30	2
12	3	30	6
14	6	30	12
4	4	30	8

Table 3.6: The routes of vehicle two.

Now each of these neighbours are checked to see if they keep the capacity constraint, as well as the vehicle road segment dependency, as it would not make sense to keep them otherwise. Then we check if they violate any of the early signs that they wont be better than the current. That is the size of their fixed cost, i.e. the time it would take to service all arcs after connecting the sub-graphs, assuming that one can always go directly from one required arc to the next, without any deadheading. If this time expenditure exceeds that of the best currently known solution, then this arc allocation will be inferior as the time usage of the route will always be greater than or equal to this fixed cost, and there is thus no need to find the route if we already know it will be worse. Another early sign we use is this index of the connectedness or dispersion of the graphs, specifically it is the number of disconnected sub-graphs divided by the number of arc, so for vehicle one in the above for example this number would be $\frac{3}{7}$ and thus it would survive as our limit is 0.5. Neighbours in violation of any of these limits are pruned away. Then we draw a number of random solutions (these are also checked for feasibility, fixed cost and dispertion) which are not bound by the current population and these are added to the neighbourhood list. The neighbourhood is then put through the process of route finding and evaluated, after which the population is updated along with the best known solution.

Allocation \Arc	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Current population	1	2	1	2	2	1	2	1	1	2	1	2	1	2
Neighbour 1	2	2	1	2	2	1	2	1	1	2	1	2	1	2
Neighbour 2	1	1	1	2	2	1	2	1	1	2	1	2	1	2
Neighbour 3	1	2	2	2	2	1	2	1	1	2	1	2	1	2
Neighbour 4	1	2	1	1	2	1	2	1	1	2	1	2	1	2
Neighbour 5	1	2	1	2	1	1	2	1	1	2	1	2	1	2
Neighbour 6	1	2	1	2	2	2	2	1	1	2	1	2	1	2
Neighbour 7	1	2	1	2	2	1	1	1	1	2	1	2	1	2
Neighbour 8	1	2	1	2	2	1	2	2	1	2	1	2	1	2
Neighbour 9	1	2	1	2	2	1	2	1	2	2	1	2	1	2
Neighbour 10	1	2	1	2	2	1	2	1	1	1	1	2	1	2
Neighbour 11	1	2	1	2	2	1	2	1	1	2	2	2	1	2
Neighbour 12	1	2	1	2	2	1	2	1	1	2	1	1	1	2
Neighbour 13	1	2	1	2	2	1	2	1	1	2	1	2	2	2
Neighbour 14	1	2	1	2	2	1	2	1	1	2	1	2	1	1

Table 3.7: Neighbourhood of the arc allocation in the example.

With this proposed solution we hope to be able to handle the more practical aspects of winter gritting introduced in Chapter 1. However to further understand our proposed solution and find potential for further development of the algorithm we need to analyse the performance.

Chapter 4

Analysis

We start the analysis with a parameter tuning test, where we try population sizes and number of mutations ranging from 1 to 10 and for each do 100 runs of the proposed algorithm which has been implemented in R. From these tests we will chose a parameter setup which we will use on data from [17]. However this data is from waste collection and thus we will disregard the waste-properties in the data and as road width is not something that has previously been included in models we will use randomized widths, drawn from a uniform distribution between 6 and 12 meters. We will then compare results from the algorithm descriped in Chapter 3 vs one only running the greedy RPP heuristic, for a range of problem instances, as there are no existing solutions to compare our results to.

4.1 Parameter tuning

The median objective values of the 100 runs of each parameter setup is shown in Table 4.1. We choose to compare medians to get a more robust choice of parameters. We clearly see a tendency towards larger populations giving better objective values. We also see that above a certain number of mutations the value of additional mutations diminishes. Similarly the median computational time is shown in Table 4.2. Here we note that the computational time is roughly multiplied by 6 from having a population of only one, to having a population size of 10. However there is no clear picture that adding more mutations has significant impact on computational time. We are now faced with a trade-off, it seems that the computational time has a level of less than a minute with population sizes below seven, after which it increases to more than 1.5 minutes, while the objective value stops decreasing as much when the population size is above five. We therefore choose a population size of six, as it seems enough to get a good objective value, while keeping computational time at a reasonable level, we choose five mutations as no gain seems to come with more mutations than that.

	1	2	3	4	5	6	7	8	9	10
1	56.12	55.50	56.00	56.00	56.00	55.50	55.75	55.62	55.75	56.00
2	56.00	55.50	55.50	54.75	55.25	55.38	54.75	55.25	55.00	54.75
3	55.38	55.25	55.00	55.25	54.25	54.00	54.75	54.75	54.00	54.75
4	55.25	54.75	54.00	54.00	54.12	54.00	53.62	54.75	53.25	54.00
5	55.25	54.75	55.00	53.25	53.25	54.00	53.25	54.75	53.62	53.25
6	54.25	54.00	54.12	53.25	53.25	53.25	53.25	53.25	53.25	53.25
7	54.25	54.50	53.25	53.25	53.25	53.25	53.25	53.25	53.25	53.25
8	53.25	54.00	53.25	53.25	53.00	53.00	53.25	53.25	53.25	53.12
9	53.25	53.62	53.25	53.25	53.00	53.12	53.00	54.00	53.00	53.00
10	53.25	53.00	53.25	53.25	53.62	53.25	53.00	53.00	53.12	53.00

Table 4.1: Median objective values, rows are population size and columns show number of mutations. We highlight values ≤ 53.25, and know from brute force that optimal objective is 52.5.

	1	2	3	4	5	6	7	8	9	10
1	0.34	0.30	0.33	0.28	0.33	0.32	0.29	0.30	0.33	0.36
2	0.47	0.45	0.46	0.45	0.48	0.42	0.49	0.46	0.53	0.55
3	0.64	0.59	0.47	0.51	0.44	0.53	0.48	0.51	0.48	0.48
4	0.56	0.60	0.60	0.63	0.63	0.59	0.65	0.54	0.58	0.59
5	0.65	0.64	0.61	0.84	0.89	0.73	1.42	1.44	1.39	1.32
6	1.13	0.72	0.79	0.74	0.69	0.77	0.72	0.78	0.79	0.78
7	0.90	1.70	1.58	1.61	1.68	1.56	1.52	1.47	1.50	1.45
8	1.54	1.36	1.45	1.45	1.49	1.69	1.52	1.55	1.53	1.66
9	1.62	1.46	1.70	1.62	1.73	1.64	1.55	1.69	1.63	1.71
10	1.69	1.79	1.71	1.86	1.73	1.89	1.84	1.66	1.76	1.79

Table 4.2: Median computational time per run in minutes, rows are population size and columns show number of mutations. We highlight times $\leq 1 minute$.

4.2 Test results

The way we do the testing is that we have three different fleets which we will use on a range of problem instances. The fleets are three subsets of an example fleet of 13 vehicles that COWI has provided. We chose these subsets to represent both trucks and tractors, they are shown in Figure 4.3, and are denoted fleet A, B, and C, for simpler representation in test comparisons. We furthermore know from COWI that the salt weighs approximately $1,200\frac{kg}{m^3}$ and that there is used $17\frac{g}{m^2}$ when spreading salt. Unfortunately, late in this project we discovered that there had been an error in the implementation, which means that the capacity constraint has not been evaluated properly, however we proceed with our analysis of the test results as we do not believe that the capacity constraint would often be the deciding factor, as the make-span objective seeks to split the workload evenly among the vehicles. We choose to use COWI's fleet data as it is readily available for us, however COWI did not have the necessarily prepared data available. Preparing such data can be a rather time consuming process and since this is not the focus of this project we choose to use the data from [17], which is a rather new dataset from 2018 from 5 areas in Denmark and thus we believe that it is comparable to what COWI's data would be like. Some of the properties of the test instances are listed in Table 4.4. For each combination of fleet, instance, and solution method we do 50 trials and show the medians, to not be affected by outliers, and compare results.

ID	vehicle	spreadwidth [m]	capacity [m^3]	service speed [kph]	deadhead speed [kph]	Fleet(s)
1	Truck	12	6	30	60	A,B,C
5	Truck	8	5	30	60	A,B,C
8	Tractor	6	5	15	30	B,C
9	Tractor	6	5	15	30	C

Table 4.3: Fleets, A is the first two vehicles, B is the first three vehicles and C is all vehicles.

Instance	Arcs	Nodes	Required arcs	Required length
1	33	26	19	1859
2	35	28	26	2519
3	106	82	76	7561
4	110	80	72	7688

Table 4.4: Test instance properties.

Because of the computational speed of the two algorithms we compare. We are unable to include massive tests, and since the one including the Frederickson algorithm is slower than the purely greedy algorithm we have fewer results to present for this algorithm. The test results are shown in Table 4.5.

We recall that the fleets are increasing in the sense that more capacity and vehicles are added from A to B to C, and thus we would expect the objective functions to be strictly decreasing, however we see an increase in each instance when we use fleet C. Similarly we would expect it to be slower, as more vehicles means a larger neighbourhood to search through, however observe the opposite, this is the case for both algorithms. These two things combined tells us that the algorithms may converge too fast and not search the solution space thoroughly enough to find the better solutions. This could be because of the random population that is generated

Instance	Fleet	Median objective greedy RPP	Median time [minutes] greedy RPP	Median objective proposed solution	Median time [minutes] proposed solution
1	A	3541	3.23	3667	21.19
1	B	2970	2.53	2803.5	29.62
1	C	4128.5	1.18	4613	4.27
2	A	4834	6.66	4873.5	50.99
2	B	3543	13.5	3530.5	92.99
2	C	5807	1.74	5878	8.27
3	A	14709	242.2		
3	B	10743.5	441.2		
3	C	24649	9.02		
4	A	15150.5	153.56		

Table 4.5: Results from tests on CARP instances from [17], with randomized widths.

in the begining of the algorithms, but when this is seen in the median we cannot explain it purely by randomness. This would argue that the neighbourhood search is not too good at searching the solution space, and that our random mutations are not enough to counter the effect of fast local convergence. Thus it serves as an argument that more emphasis should be put into finding a searching algorithm that can more thoroughly search the solution space, or be smarter at constructing good guesses. Furthermore we would expect that the proposed solution would be better than the purely greedy algorithm, but they are in-fact roughly equal in objective value, with the proposed solution much slower. This is partly due to the Frederickson heuristic being more time consuming, but since the search algorithm has not been bound by a time-limit, this does not explain them having roughly equal objective values. This serves as a second argument that the search of the solution space should be prioritized further, and that computational time should not be prioritized towards a good RPP solution if the time could instead be spent on a better, or more thorough searching of the solution space.

As for whether these computational times are reasonable in a practical sense we believe that they are. While the test instances we include here are not the most extreme one could imagine, we see that for roughly 100 arcs connected by 80 nodes with 75% of them requiring service, we compute a result in less than one day, which should be sufficient to allow for prototyping changes in the overall system, though it would not be fast enough for day-to-day changes in which roads require service, which could often be the case for winter gritting. However we also note that these tests are carried out on implementation in R which is not famous for its great speed, so one would expect it to be implemented in C++ if used commercially.

Chapter 5

Conclusion

The aim of this project was to provide a solution to the real-life Capacitated Arc Routing Problem (CARP), faced when winter gritting with a heterogeneous fleet of gritting vehicles. We located a number of aspects that should be included in cooperation with COWI who faces this type of problem every year. We found that there was especially two gaps in the literature and chose to aim at closing one of them. Namely the problem that some vehicles needs to service certain road segments more than once in order to service it fully. We then went on to propose a solution algorithm which can handle this type of problem, without loosing the ability to handle the other aspects that has already been covered in existing research. The solution is a heuristic which does a neighbourhood search around a population of best known solutions, which is iteratively updated and forms a new neighbourhood, until no improvements are found. Furthermore, a number of randomly mutated potential solutions are added to the neighbourhood at each iteration to try to counter convergence towards poor quality local optima. It facilitates the use of well known heuristic, the Frederickson heuristic, to solve the underlying Rural Postman Problems (RPP), which in turn enables an evaluation of the objective function. The objective of the problem is the make-span, as it was determined in collaboration with COWI to be the most realistic objective to minimize. The algorithm has two parameters, the population size and the number of random mutations, was tuned with an example case, and chosen to be five mutations and a population of size six.

This setup of the proposed solution algorithm is then tested on a set of test instances from the literature, with randomly simulated road widths, as this parameter has not yet been introduced to available test data for this type of problem. This also means that we have no existing solution method to compare the quality of our solution against, which is why we do tests for both the proposed solution and a greedy algorithm which does not include the Frederickson heuristic. We note first that the method works however the results of the tests are not unambiguous, as the objective value increases in some instances where resources are added. Computational time decreases in the same instances, where you would expect the problem to become more complex and thus take more time. There can be a number of explanations for this. It could be that there is some underlying quality of our specific test instances that makes this happen, however this seems less likely than the previous two explanations and further testing would be required to state that this is the case. More likely it could be that there are not enough random mutations added, in order to counter fast local convergence in complex cases, or that the neighbourhood search is not very well suited to search through the type of solution space presented in this problem.

We also see from the tests that the Frederickson heuristic does not add much quality when compared to the tests we do with only a greedy algorithm for the RPP. So while the Frederickson heuristic is significantly slower than the greedy algorithm, it provides roughly the same results. This indicates that further prioritization should be placed on the searching part of the algorithm instead of improving the solver for the underlying RPP, especially if the computational time is critical. Finally the algorithm proves reasonable in computational time, however we note that if it was to be used in a commercial setting where time is an important factor, it should be implemented in a faster coding language than R, e.g. C++.

5.1 Further work

If the project was to continue, first step would be further testing to know more about why we see the increase in objective value, when adding resources. With these tests one should then try to improve the searching algorithm which allocates the arcs to certain vehicles. We think this should be prioritized above finding better (shorter) routes to cover the allocated arcs, as we have seen a simple greedy algorithm do this part of the job in the algorithm as well as a well known and tested heuristic. Furthermore of course if additional time was allocated to this project one should also include the change of travelling speed that happens once an arc has been gritted, as this will also have some affect on the optimality of a given solution. However along with this being less impact-full in our minds, it is an immensely complicated thing to evaluate, as some vehicles will deadhead road segments that has been serviced by other vehicles, which makes for an inter-vehicle-dependant evaluation, as no make-span for any vehicle can be calculated knowing only its own route.

Bibliography

[1] L. Euler, Solutio problematis ad geometriam situs pertinentis, Commentarii academiae scientiarum Petropolitanae (1741) 128–140.

[2] M. Kwan, Graphic programming using odd or even points, Chinese Math 1 (1962).

[3] C. Orloff, A fundamental problem in vehicle routing, Networks 4 (1) (1974) 35–64.

[4] B. L. Golden, R. T. Wong, Capacitated arc routing problems, Networks 11 (3) (1981) 305–315.

[5] S. Wøhlk, A decade of capacitated arc routing, in: The vehicle routing problem: latest advances and new challenges, Springer, 2008, pp. 29–48.

[6] About COWI (accessed October 23rd, 2019).
URL `https://www.cowi.com/about`

[7] M. C. Mourão, L. S. Pinto, An updated annotated bibliography on arc routing problems, Networks 70 (3) (2017) 144–194.

[8] N. Perrier, A. Langevin, C.-A. Amaya, Vehicle routing for urban snow plowing operations, Transportation Science 42 (1) (2008) 44–56.

[9] B. Dussault, B. Golden, C. Groër, E. Wasil, Plowing with precedence: A variant of the windy postman problem, Computers & Operations Research 40 (4) (2013) 1047–1059.

[10] T. Liu, Z. Jiang, N. Geng, A genetic local search algorithm for the multi-depot heterogeneous fleet capacitated arc routing problem, Flexible Services and Manufacturing Journal 26 (4) (2014) 540–564.

[11] A. M. Rodrigues, J. Soeiro Ferreira, Waste collection routing—limited multiple landfills and heterogeneous fleet, Networks 65 (2) (2015) 155–165.

[12] S. Høyland, J. A. Krogstad, The snow grooming routing problem with multiple depots and heterogenous fleet-comparing an exact solution approach with localsolver, Master's thesis, NTNU (2019).

[13] S. Wøhlk, G. Laporte, A fast heuristic for large-scale capacitated arc routing problems, Journal of the Operational Research Society 69 (12) (2018) 1877–1887.

[14] K. Holmberg, Heuristics for the rural postman problem, Computers & Operations Research 37 (5) (2010) 981–990.

[15] G. N. Frederickson, Approximation algorithms for some postman problems, Journal of the ACM (JACM) 26 (3) (1979) 538–554.

[16] K. H. Rosen, K. Krithivasan, Discrete mathematics and its applications: with combinatorics and graph theory, Tata McGraw-Hill Education, 2012.

[17] L. Kiilerich, S. Wøhlk, New large-scale data instances for carp and new variations of carp, INFOR: Information Systems and Operational Research 56 (1) (2018) 1–32.