Journal on Vehicle Routing Algorithms (2018) 1:105–119
https://doi.org/10.1007/s41604-018-0008-3

REGULAR PAPER

Heuristics for the weighted *k*-rural postman problem with applications to urban snow removal

Kaj Holmberg[1]

Received: 3 April 2017 / Accepted: 7 February 2018 / Published online: 7 March 2018

Abstract
We describe a weighted version of the k-Chinese and k-rural postman problem that occurs in the context of snow removal. The problem concerns the questions of which vehicle shall take care of each link and how the vehicles shall travel between links. We also consider different numbers of vehicles, in view of a fixed cost for each vehicle. We describe and discuss heuristic solution approaches, based on usable substructures, such as heuristics for rural postman problems, meta-heuristics, k-means clustering and local search improvements by moving cycles. The methods have been implemented and tested on real life examples.

Keywords k-Chinese postman problem · Rural postman problem · Heuristics · Clustering

1 Introduction

Snow removal can be an important and costly issue, in certain places at certain times. The yearly amounts of snow has recently varied much in several countries. Some years there is a lot, other years hardly anything. Hence, the demand of snow removal varies a lot. Snow removal can be quite difficult and expensive, if there is very much snow. Winters without snow make it difficult to motivate keeping a large fleet of snow removal vehicles available. Complaints from the public about snow removal are common. A conclusion is that one cannot expect to use the plans from last year. New situations require new plans, which require planning tools.

Let us describe the planning situation (in Sweden). A city is divided into areas, and for each area, a contractor is hired. As the contractors work independently, such an area gives natural borders for the optimization. Each contractor has a number of vehicles/drivers that together will remove the snow from all streets in the area. Each driver is allocated a set of streets, and in practice is often free to plan the tour by himself. Good tours are often based on years of experience, but this may fail when new drivers are hired, or if there has been very little snow for many years. Another aspect that might not be optimal is that the drivers' designated areas are for practical reasons usually as separate as possible, since their tours are not coordinated.

The treatment of a street includes many details, such as several sweeps, depending on the width of the street, additional clearing at crossings, and even considerations of the fact that it might take a longer time turning a vehicle around than continuing straight ahead. Such details are taken into account in a huge MIP-model presented in the technical report [15]. Unfortunately, that MIP-model is practically impossible to solve. In [15] we also present a number of relaxations of the model, which may be used to get bounds on the optimal objective function, but do not give feasible solutions to the problem.

Let us now describe the problem to be solved here. We consider the area assigned to one contractor, and focus on the question how to allocate the streets to the vehicles. To evaluate an allocation, we need to find tours for the vehicles. However, there may in practice always be small disturbances (parked cars, more or less snow than expected, etc.) that lead to deviations from the planned tours. Therefore, we see the allocation of streets to vehicles as the most important part of a solution.

We will do the optimization in stages, first find an allocation of streets to vehicles, and then find good tours for the vehicles. Obviously, we need to iterate and modify the allocation based on the costs of the tours.

✉ Kaj Holmberg
kaj.holmberg@liu.se

[1] Department of Mathematics, Linköping Institute of Technology, SE-581 83 Linköping, Sweden

When a good allocation is finally at hand, a tour taking all details into account will be produced. That step is described in [19]. Therefore, we here omit many details, and approximate the work needed to clear a street by one operation, since the purpose of the tour is mainly to evaluate the allocation.

We assume that the vehicles are identical, and do not use depots, since we do not know where the vehicles will start. In practice, this means that a vehicle can enter the tour at any node.

Finding data for the problem can be an issue. The street network and all link characteristics can nowadays be found in different databases. In this paper, we use data from OpenStreetMap. In [18], we give more details about what data there is and how to extract the data to a useful form.

Times for doing tasks are known approximately, based on an average speed. The operators will in the future be required to equip each vehicle with a GPS. This gives data from which we may extract times for all tasks. Therefore, task times will be available with better accuracy. Analyzing the GPS tracks to extract that information will require map matching, which is described in [17].

We can solve the problem for different numbers of vehicles, as a way to dimension the fleet of vehicles. Clearly, it is faster to let several vehicles work in parallel, but there is a fixed cost for each vehicle that does not depend on how much the vehicle is used, only on the fact that it is available.

What kind of optimization problem is this? If there was only one vehicle to do all snow removal, the problem would be reduced to a Chinese postman problem, which simply is to find a shortest round trip that covers all links in a graph.

Related problems are the following. The rural postman problem is to find a shortest round trip that covers certain links. This corresponds to one vehicle clearing a subset of the links. The k-Chinese postman problem is to find k shortest round trips that covers all links, which corresponds to k vehicles clearing all links. The k-rural Postman Problem is to find k shortest round trips that covers certain links, which corresponds to k vehicles clearing a subset of the links.

An important question is the objective function. We could minimize the makespan of the whole operation, i.e., minimize the maximal time of a vehicle. This leads to the min-max k-Chinese postman problem. We could also minimize total cost for the clearing, i.e., minimize the sum of the times for the vehicles. This leads to the original k-Chinese postman problem. Minimizing total time will also minimize total distance traveled when the vehicles are not clearing, which in principle minimizes the pollution caused by the vehicles. The best objective function is, we believe, a weighted combination of those above.

Our snow removal planning problem therefore turns out to be a new version of the k-Chinese postman problem. More precisely, it is a weighted combination of the original k-Chinese postman problem, where the total distance is minimized, and the min-max k-Chinese postman problem, where the maximal distance traveled by a vehicle is minimized. The two objectives are here simply added together with certain weights. In practical terms, it is a compromise between minimizing the cost for all travel (and equipment) and minimizing the time until all snow is removed.

If there are links that we do not need to treat, we have the k-rural postman problem. There could be links where the snow already has been removed, or where the responsibility of snow removal lies elsewhere.

The computational tests in this paper have been done with the two tools, Snowplan and Vineopt (implemented by ourselves). In Fig. 1, we show a preliminary run with Snowplan, and in Fig. 2, we show how the network and allocation are brought into Vineopt. More details about these tools are given later.

Finally, we wish to point out that the techniques used in this paper can be applied to many other multi-vehicle arc routing applications, not only snow removal.

2 Survey

The Chinese postman problem is a well-known problem where each link in a graph needs to be covered. It can be solved optimally and polynomially by a well-known method, [9,10]. The rural postman problem is a similar problem, where, however, only a subset of the links need to be covered. This problem is NP-hard, but there exist efficient heuristics, [5,6,11,13,14].

The k-Chinese postman problem is treated in the following publications. In [24], complexity results for different versions of the Chinese postman problem are generalized to versions of k-Chinese postman problems. In [1,2], heuristics and metaheuristics are presented for the min-max k-Chinese postman problem. In [23], new variants of the k-Chinese postman problem are presented, together with heuristics for the k-Chinese postman problem. In [3] the min-max k-vehicles windy rural postman is treated. General references for arc routing are [7,8].

Winter road maintenance is treated in the following papers. In [4], the snow disposal assignment problem is modeled as a multiresource generalized assignment problem, and a two-phase heuristic is developed. The paper [21] presents models and methods for partitioning a city into sectors for snow disposal operations, and for assigning the sectors to disposal sites. The heuristics include a penalty-based assignment phase, a two-opt exchange and a districting algorithm of savings-type. The thesis [31] proposes a real-time scheduling model for winter road maintenance operations as a large MIP-model, which is solved by heuristics based on iterative solving a simplified problem with standard software. This work is continued in [12].

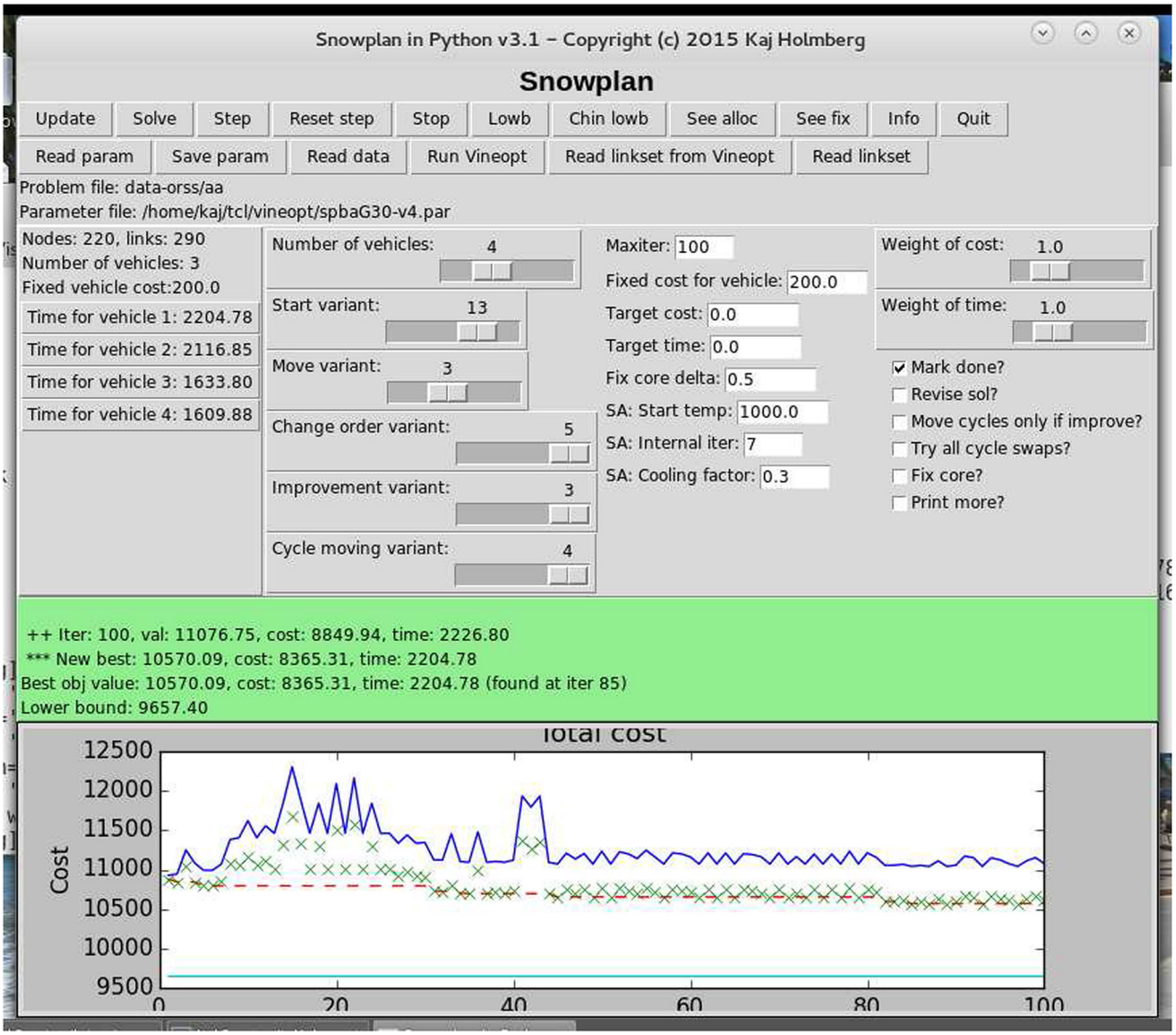

Fig. 1 Snowplan: Åtvidaberg, G100, $q = 4$

Rural snow removal is discussed in [20,30], where the focus lies on finding a set of periodic routing paths during a continued snowfall. Heuristics, including simulated annealing, are used.

A comprehensive survey of problems related to winter road maintenance is found in papers [26], dealing with system design for spreading and plowing, [27], dealing with system design for snow disposal, [28], dealing with vehicle routing and depot location for spreading, and [29], dealing with vehicle routing for snow plowing and disposal. Many optimization models and solution methodologies are presented and categorized. The paper [25] presents a model for the routing of vehicles for snow plowing operations in urban areas. The model is large, and two heuristic approaches are used. In the first, several routes are constructed in parallel by sequentially solving a multiple vehicle rural postman problem with vehicle road segment dependencies, turn restrictions, and load balancing. The second uses the approach of cluster-first route-second, and first determines a partition of the arcs into clusters, with similar work load, and then for each cluster solves a hierarchical rural postman problem with class upgrading possibilities, vehicle road segment dependencies, and turn restrictions. This paper is very interesting for us, but the work is aimed at solving the problem once each winter season. We wish to be able to solve our problem each day (if necessary).

3 Notation

An example of a model for the min–max k-rural postman problem is given in [3]. Here we do not plan to use MIP-

Fig. 2 1045: Vadstena, Vineopt, $q = 6$

codes for solving the problem, so we will not go into all the details of such a model. However, some notation will help in the description of the heuristic methods.

We will use the following indices: $k \in Q$: the set of vehicles, $j \in L$: the set of links (streets), and $i \in N$: the set of nodes (points, crossings). We will use $q = |Q|$, $n = |N|$ and $m = |L|$. The links are assumed to be undirected, i.e., the combined operation discussed previously can be done in either direction. Link j has length l_j, which is obtained from the digital maps.

The set of links that shall be cleared of snow are denoted $L^C \subseteq L$. Often $L^C = L$, i.e., all links in the network need to be cleared, but sometimes there are links that will be cleared by another operator, or by other types of vehicles, but still are available to use for transport of our vehicles. As will be clear later, for our method it will not matter much if $L^C = L$ or not.

The value of q, the number of vehicles, may be fixed, but we also consider the case when it can be varied. We will not make a model where this parameter is a variable, but will rather do a parametric analysis, i.e., solve the problem for different values of q and compare the results, as there are only a few interesting values.

We also need the following data:

t_j : the time needed for a vehicle to clear link j.
d_j : the time needed for driving a vehicle along link j (while not clearing).

We will consider the average speed to be v^A m/s, so it will take $d_j = l_j/v^A$ seconds to drive the whole street j. The time to clean a road is estimated in average to be p times the length of the road divided by the speed. A value of $p = 2$ is reasonable for an ordinary street. This time includes two or three sweeps (middle, right side, left side) as well as cleaning

of turning space and/or crossroads. All of this is seen as one operation that starts in one end of the street and finishes in the other end. For narrow streets or bicycle paths, $p = 1$ would be more appropriate.

It then takes $t_j = pd_j = pl_j/v^A$ seconds to clean road j. We will assume that the cost for driving a vehicle is proportional to the time it is driven, so letting c^D be the cost for driving one second, the cost for cleaning link j is $c_j = c^D t_j = c^D pl_j/v^A$.

The total time needed for clearing is $t^{TOT} = \sum_{j \in L^C} t_j$, which is a constant. The optimization will only concern the rest of the solution, which is traveling around with the vehicles without removing snow.

We will use f as the fixed cost for using a vehicle, when considering the number of vehicles.

We also need the following:

z^R : the time when all vehicles are ready.
z_k^U : the total time vehicle k is used.
z_k^C : the total time vehicle k is used for clearing snow.
z_k^L : the total time vehicle k is used for transportation.

Some relevant objective functions are the following:

Minimize final end time: min z^R
(the min-max k-Chinese/rural postman problem)

Minimize sum of total times for vehicles: min $\sum_k z_k^U$
(the original k-Chinese/rural postman problem)

Minimize sum of transport times: min $\sum_k z_k^L$

As there is a fixed proportion between time and cost (c^D), the last two objective functions above could just as well be regarded as minimizing the cost. They are often accompanied by a constraint on z^R (the snow must be removed within a given time frame).

In this paper, we will use an objective function that is a weighted sum of the maximal time for a vehicle and the total cost for the whole operation.

$$\min w^A z^R + w^C \left(c^D \sum_k z_k^U + fq \right),$$

where w^A and w^C are the weights reflecting the importance of time relative to cost. If w^A is large, it is important to finish early, i.e., to minimize the total time until all links are cleared, while if w^C is large, it is more important to minimize the total costs for the whole operation.

When we solve the problem, fq will be constant, but we may solve the problem for a couple of different values of q, and compare the results, and then this term will be important.

4 Lower bounds

We will later describe heuristics for finding feasible solutions. This yields upper bounds on the optimal objective function value. It can then be useful to obtain some lower bounds on the optimal objective function value, by solving a relaxation of the problem.

We can then compare the upper and lower bounds to estimate the possible potential for improvement. If we are very lucky, the lower bound may be close enough to the upper bound, so that we may conclude that the solution is close enough to the optimum.

A possible use for a relaxation is to construct a branch-and-bound method around it. By branching, the lower bound will get closer to the upper bound, and finally indicate optimality. A possible binary branching is to fix that vehicle k should do task j in one branch and not in the other. Another possibility is to create q branches for a certain link j, where vehicle k does task j in branch k.

The simplest lower bound, denoted by l^0, is obtained by dividing the total work that needs to be done by the number vehicles. Here all transportation is omitted and the need for forming connected cycles is ignored.

A slightly better lower bound is to find the cost of one Chinese postman tour, z^P, i.e., the cost for one vehicle to do all, and divide by the number of vehicles. This yields the bound $l^1 = w^A t^1/q + w^C(c^D t^1 + fq)$, where $t^1 = z^E/p + t^{TOT}$, since the extra work, $z^E = z^P - t^{TOT}$, is only transportation.

The time for obtaining l^0 is negligible, while the time needed to obtain l^1 is somewhat longer, but still much shorter than what is needed for solving the problem, i.e., finding a good feasible solution for $q > 1$. In our framework, we choose to calculate l^1.

5 Solution methods

We will focus on heuristic solution methods, such as constructive heuristics, local search and metaheuristics. The main parts are the following. First, we make an initial allocation, i.e., divide the links among the vehicles. Then we find a feasible solution, i.e., a route for each vehicle. After this, we try to improve the routes by local search, and by metaheuristics.

A high-level algorithm for the method is given as Algorithm 1. Details will be specified afterwards.

The total cost, which is the sum of the costs for each vehicle's tour plus the fixed costs for vehicle, is computed at each iteration. Moreover, the time for the solution is computed, which is the maximal over the vehicles' times. The objective function value is then computed as a weighted sum of the cost and the time.

Simulated annealing is used for moving to a new allocation, i.e., better allocations are always accepted and worse allocations are accepted with a certain probability.

Steps 3, 9 and 10 can be done in various ways, as described later. We also have to choose which of the optional steps

Algorithm 1 High-level algorithm

1: Read the network.
2: Construct a starting order for the vehicles.
3: Construct an initial allocation of links to vehicles.
4: Solve the rural postman problem for each vehicle.
5: Optional: Make a vehicle clean all uncleaned links passed by it, even if they were allocated to another vehicle.
6: Evaluate the objective function.
7: Optional: Improve the solution by moving cycles from one tour to another one.
8: Optional: Change the allocation to the current solution.
9: Change the order for the vehicles.
10: Change the allocation of links to vehicles.
11: Repeat steps 4–10.
12: Save the allocation of links to vehicles.

5, 7 and 8 to do. These choices are governed by parameters that will be specified below. (Some possibilities that in preliminary tests in [16] were found to be inferior are removed.)

Two important vectors are o, which is the order in which the vehicles are considered, and κ, which states which vehicle should clean each link. More precisely o_i is the vehicle (index) treated as the i-th one in the loop, and κ_j is the vehicle (index) that is assigned to link j.

5.1 Starting allocation

It can be very important which starting allocation is used. One possible quick use of the method is to use only the starting allocation, without subsequent tries to improve it. It is controlled by the parameter SV.

0: Start from the current allocation.
1: Create a random allocation.
2: Read a link set from file and use as a starting allocation.
3: Sort in latitude for starting node.
4: Sort by distance from the origin to starting node.
5: Use *k-means*, with boundary links allocated to the highest number.
6: Use *k-means*, with boundary links allocated to the lowest number.
7: Use *k-means*, with boundary links randomly allocated.
8: Use *k-means*, with boundary links allocated to minimize distance to center.

Using SV $= 0$, we can repeatedly try to improve the current solution, while resetting the simulated annealing temperature between the runs. The case SV $= 2$ enables starting from a previously constructed allocation. This can be used iteratively to continue with a previous allocation that has been improved by other means (in Vineopt). For SV $=$ 3–4, the links are sorted based on position, and the allocation is made for one vehicle at a time, picking the closest links in order. It can be seen as a kind of (crude) geographical clustering. For SV $=$ 5–8, the nodes are clustered in q groups, see Sect. 5.2, thereby inducing an allocation of the links for the vehicles. The difference is how boundary links, i.e., links with the end nodes in different clusters, are treated.

5.2 *k*-means clustering

Given the coordinates of the nodes N, the k-means clustering problem means finding k centers and allocating each node to a center, so that the sum of distances from each node to its allocated center is minimal. The problem is *NP*-hard, but efficient heuristics exist, see for example [22].

This can be used for our problem with $k = q$, i.e., to create one cluster for each vehicle. This gives a partitioning of the nodes to the vehicles. Since we are more interested in the links, we must do the following. If both the starting and ending nodes of a link belongs to the same cluster, the link is allocated to that cluster. However, if the starting node and the ending node of a link belongs to different clusters, we must decide to which cluster of the two the link should belong. In our computational test, we have tried some simple heuristic ways, namely to allocate the link to the cluster that has the highest index of these two, or to the lowest index, or randomly to one of the two clusters.

We have also tried the following. If link j has its two end nodes in different clusters, $s_j \in A$ and $e_j \in B$, we calculate the "cost" of letting each cluster cover the other node, as the distance from each cluster center to the other endpoint of the link. If this distance is large, the vehicle has to travel far from its center to include the link. Obviously, this is no exact measure at all, but it may give some indication. We then choose the allocation that minimizes this measure.

Let $d(A, s_j)$ denote the distance between center of cluster A and node s_j, and $d(B, e_j)$ the distance between center of cluster B and node e_j. If link j is allocated to cluster A (i.e., node e_j is covered by cluster A), we get the distance $d(A, s_j) + l_j$, and if link j is allocated to cluster B (i.e., node s_j is covered by cluster B), we get the distance $d(B, e_j) + l_j$. Obviously, if $d(A, s_j) < d(B, e_j)$, then the "cost" is least if we choose the first alternative. Thus, if $d(A, s_j) < d(B, e_j)$, then link j is allocated to cluster A, and if $d(A, s_j) > d(B, e_j)$, then link j is allocated to cluster B.

In this paper, we will use "vehicle" and "cluster" interchangeably, both indexed with k.

Using clustering this way will produce allocations where vehicles rarely go into each other clusters, which might seem good. However, in Fig. 3, we give two cycles, one dashed, with lengths on the paths. Swapping the two vehicles on the shorter paths as shown to the right in the figure, makes the longest cycle shorter, so it improves the solution.

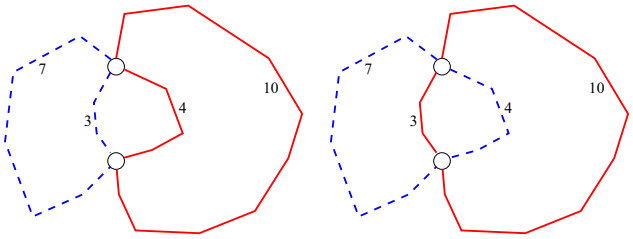

Fig. 3 Changing allocation

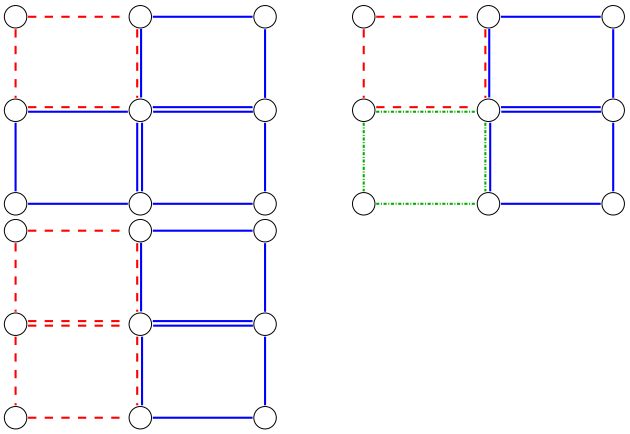

Fig. 4 Moving cycles

Thus, we see that it may be beneficial to let vehicles cross paths.

5.3 Finding a tour for one vehicle

We recall that routing a vehicle on a given set of undirected links is a rural postman problem, and efficient heuristics for this problem are presented in [14]. The set of required links for vehicle k is simply those links that have $\kappa_j = k$.

A practical aspect (omitted in the problem formulation) is that it may take a longer time to turn a vehicle around than to continue around the block. However, we can do a slight effort to avoid turning as follows.

When solving a rural postman problem, additional arcs/edges are added to form a graph where an Euler tour exists. The procedure ends by finding an Euler tour in the extended graph. In this stage, the Euler tour is never unique, as it is a collection of subtours that can be taken in any order. All Euler tours are equally good, so we may choose any one of them. Then it is possible to avoid a U-turn as much as possible, by simply avoiding going back to the recently visited node if possible.

5.4 Improve solutions

As an example, in Fig. 4, we first give two tours, one long (blue) and one short dashed (red). Both vehicles pass the middle node, so one may move the dash-dotted cycle (green) in the second graph from the long to the short, without the need for any vehicle to move to any other node. The total distance is unchanged by this, but the longer cycle becomes shorter, and the shorter cycle becomes longer, so the maximal time is decreased.

We thus try to improve the solution by finding a cycle in a tour, checking if the cycle contains a node in another tour, and then moving the cycle to that tour. This does not change the total cost for the solution, as precisely the same subtours are used. However, moving a subtour from one vehicle to another may decrease the final time. It is controlled by the parameter IV:

0: Do not move cycles.
1: Move cycles, try to move a cycle from an expensive tour to a cheap one.
2: As 1, try all combinations.

5.5 Mark links done

When the tour of the first vehicle is found, one usually finds that the vehicle passes links that are not allocated to be cleared by that vehicle.

Then it is possible to "disobey" the allocation, and let that vehicle clear all links it passes. These links are then done, and later vehicles need not pass them. It is not certain that this is a good idea, since it could mean that the first vehicle does a lot of work while the following do less and less. (Here the difference between c and d matters, i.e., how much faster a vehicle drives without clearing. It is probably a better idea if this difference is small.). It could, however, be worth to try.

Consider the example in Fig. 5. First, we give the allocation for two vehicles, one dashed, and then the cycle of

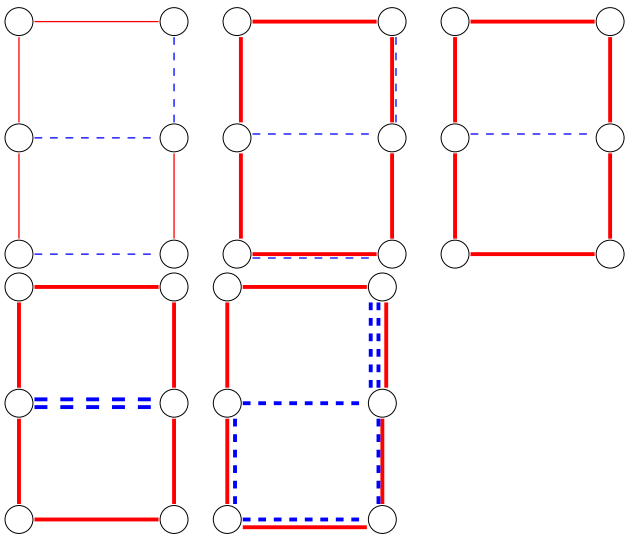

Fig. 5 Changing allocation

the first vehicle as a thicker line. Then we change the allocation of the dashed links that the first vehicle passes to that vehicle. After this, the dashed cycle only need to cover the middle link. Assuming length 1 of each link, the first cycle has length 6, and the second cycle length 2, which sums up to 8. In the last picture, we show how long the second tour would be if the allocation was not changed. Then the second cycle would get length 6, so the sum of lengths would be 12.

The method first finds a tour for the first vehicle, which passes all links that are allocated for the vehicle. If MD = 1, all links passed by the vehicle are marked as done, i.e., these links do not need to be treated by other vehicles. If we change the order in which tours for vehicles are decided, a different solution could be generated.

Therefore, the vector o can be changed to get different solutions. There are several different approaches to (randomly) change o, see below.

Step 8 in the algorithm is only interesting if step 5 is used. It decides if the changed allocation should be saved for later iterations, RS $= 1$, or if the old allocation (which was not followed) should be kept, RS $= 0$.

5.6 Change order of vehicles

When finding the routes for the vehicles, we solve a rural postman problem for each vehicle, and one question is the order in which the vehicles are handled, as discussed in the previous section. Therefore, we consider changing the order of the vehicles. We use the following possibilities, controlled by parameter ChV.

0: No change of o.
1: Swap two random elements in the vector o.
2: Create a completely new order.
3: Swap the most and least used vehicles.

For ChV $= 3$, we sum up the number of links allocated to each vehicle, and switch the positions of the vehicle that has the most links allocated to it and the vehicles that has the least. If we are using MD $= 1$, this often means swapping places between the first and the last vehicle, and will make a big difference for the following tours.

5.7 Change allocation

We use simulated annealing to try to improve the allocation of vehicles to links. The neighborhood is defined by either moving a link from one vehicle to another, or swapping links between two vehicles. The changes can easily be made by explicitly working with κ_j. A move means changing one value. A swap can be seen as changing places of two elements in the vector.

The changes are controlled by the parameter MV.

0: No change.
1: Swap two random links.
2: Move a link from a vehicle that does most to a vehicle that does least.

For MV $= 2$, the number of links allocated to each vehicle is calculated, and a link is moved from the vehicle that has the highest number of links to the vehicle that has the least number of links.

In the algorithm for simulated annealing, we use the following parameters: MI, number of main iterations, ST, starting temperature, II, number of iterations with fixed temperature, RC, cooling factor. One iteration of the main algorithm is given in algorithm 2. This is just to give the structure of the method, as we cannot go into all details.

6 Computational details

6.1 Implementational details

The implementation of Snowplan is done in Python, using Numpy and Scipy (providing the code *kmeans*). The tests were run on an Acer Aspire X3 X3995 3.4GHz, running Linux, Fedora Core 22. The machine has four CPUs, but only one was used in the test runs. For visual aid, Tkinter and networkx are used. For solving the rural postman problems, we use a heuristic method described in [14], implemented in C.

We also use the graphic network optimization tool Vineopt, written by Kaj Holmberg in Tcl/Tk. It has the ability to show networks and allow changes in them, as well as solve various optimization problems in graphs, especially the Chinese postman problem (optimally) and the rural postman problem (heuristically).

Vineopt can visualize solutions and link sets (in different colors), over a map background obtained from OpenStreetMap. It also allows manual changes of link sets, representing the allocations.

6.2 Input data

In our computational tests, we consider the average speed to be $v^A = 7.2$ km/h, which is 2 m/s. The time to clean a road is estimated in average to be twice the length of the road divided by the speed, i.e., we use $p = 2$. With these assumptions, the operation takes exactly one second per meter, i.e., it takes l_j seconds to clean street j if it has length l_j. We also give one second the cost of one, so times will be equal to costs. We assume that the fixed cost is 200 for each vehicle. Therefore, the total cost for a solution is the sum of the costs for each

Algorithm 2 One main iteration

```
1: v ← 0
2: κ̂ ← κ                                          ▷ current solution
3: t^max ← 0
4: CHANGEALLOC(tour)                               ▷ change allocation
5: CHANGEORDER                                     ▷ change order
6: ldone ← 0                                       ▷ links that are done
7: for i = 1, q do                                 ▷ loop over vehicles
8:    v_j ← o[i]
9:    for j = 1, m do                              ▷ construct req for vehicle
10:      if κ[j] = v_j and ldone[j] = 0 then
11:         req[j] ← 1
12:      else
13:         req[j] ← 0
14:   WRITERUPIN(v_j)                              ▷ write indata file for rup
15:   RUNRUP                                       ▷ solve rup
16:   tourv ← READRUPOUT(v_j)
17:   obj ← CALCTOURCOST(tourv)
18:   tour[v_j] ← tourv
19:   if MD then                                   ▷ mark passed links done
20:      AREDONE(tourv,v_j)
21:   v ← v + obj + f
22:   if obj > t^max then
23:      t^max ← obj                               ▷ new max time
24: v̄ ← w^C v + w^A t^max
25: if v̄ < v^best or (v̄ = v^best and t^max < t^best) then
26:   v^best ← v̄
27:   t^best ← t^max
28:   if MD then
29:      κ^best ←REVFROMTOUR(κ,tour)
30:   else
31:      κ^best ← κ
32:   besttour ← tour
33: if IV > 0 then                                 ▷ improve solution
34:   t̂, v̄,tour ← IMPROVESOL(tour)
35: else
36:   t̂ ← t^max
37: if RS then
38:   κ ← REVFROMTOUR(κ,tour)                      ▷ update allocation
39: if v̄ < ṽ then                                  ▷ if better go there
40:   ṽ ← v̄
41:   κ̂ ← κ
42:   if v̄ < v^best or (v̄ = v^best and t̂ < t^best) then
43:      tobj ← (v̄ − w^A t̂)/w^C
44:      v^best ← v̄
45:      t^best ← t̂
46:      if MD then
47:         κ^best ← REVFROMTOUR(κ,tour)
48:      else
49:         κ^best ← κ
50:      besttour ← tour
51: else                                           ▷ or maybe save κ anyway?
52:   δ ← v̄ − ṽ
53:   p ← exp(−δ/T)
54:   if random ≤ p then
55:      ṽ ← v̄
56:      κ̂ ← κ
57:   else                                         ▷ back
58:      κ ← κ̂
59: ii ← ii + 1
60: if ii > II then                                ▷ change temperature
61:   T ← RC ∗ T
62:   ii ← 1
```

vehicle plus number of vehicles times 200. The total objective function value is here the sum of the maximal time and the total cost, i.e., $w^A = w^C = 1$.

7 Computational tests and evaluation

7.1 Test problems

The methods have been tested on several different problem instances, based on OpenStreetMap-data, obtained from the Internet. See [18] for details of the data extraction procedure. The networks cover smaller cities around Linköping, and also different parts of Linköping.

A certain filter was applied to the OSM-data, leaving only streets (highways) with the following label: "motorway", "trunk", "primary", "secondary" and "tertiary", but leaving out "road", "cycleway", "living_street", "pedestrian", "footway", "path" and "residential". These instances are thus based on real life networks, with real life distances, but with certain links removed, and nodes with degree two eliminated by simply adding the two adjacent links into one, summing up the costs. Therefore, the curvature of the link is not visible, but the distance is correct.

We have, for practical reasons, divided the set into three parts, depending on the size. The first part contains networks with less than 200 nodes. They are rather small cities or areas, and give rather easy problems. The second part contains networks with between 200 and 400 nodes. These problems are a bit more time consuming to solve. The third part contains the most difficult problems, networks with more than 400 nodes.

In Table 1, we give the number of nodes and links, the total distance of all the links (i.e., the total distance to be cleared from snow), the average node degree, and the number of inhabitants for the cities represented in the test set. For the examples based on parts of Linköping, no number of inhabitants can be specified. (The whole of Linköping has around 140,000 inhabitants.)

Pictures of some of the networks are given in Figs. 6, 7, 8 and 9.

7.2 Preliminary tests

In Fig. 1, we show a preliminary run with Snowplan. In the graph at the bottom of the picture, the blue line denotes the actual objective function value for each iteration, the green cross denotes the objective function value after the improvement procedure, and the dashed red line denotes the best objective function value obtained up to that iteration. The light blue horizontal line denotes the lower bound.

In the top of the picture, one can see the parameters that are possible to change, and their current values. At the top

Table 1 City instances, sorted in three groups

Name	Nodes	Links	Total dist	Deg	Inhab
atvid-s	68	95	2028	1.40	–
brokind	179	199	2664	1.11	502
colonia	24	36	131	1.50	–
ekangen	149	184	2629	1.23	2037
mantorp-c	122	145	2243	1.19	–
mantorp	185	226	4152	1.22	3671
rimforsa-o	94	109	1474	1.16	–
skanninge-c	122	152	2292	1.25	–
skanninge-n	62	80	974	1.29	–
skanninge	127	158	2322	1.24	3140
studentryd-e	76	104	543	1.37	–
sturefors	183	212	3283	1.16	2229
valla-liu	160	213	1183	1.33	–
vikingstad	178	213	3390	1.20	2096
atvid	220	290	5940	1.32	6859
borensberg	211	257	4994	1.22	2886
kisa	311	359	4541	1.15	3687
liu	291	394	2713	1.35	–
rimforsa	233	262	2837	1.12	2238
ryd1	382	508	2200	1.33	–
studentryd-int	235	316	1494	1.34	–
studentryd	387	519	2920	1.34	–
linghem	836	1129	6624	1.35	518
ljungsbro	812	1024	14, 013	1.26	6620
malmslatt	489	626	6542	1.28	5214
mjolby	419	524	11, 894	1.25	12,245
ryd-m	1107	1495	10, 910	1.35	–
ryd	952	1291	7903	1.36	–
ryd2	987	1337	9174	1.35	–
soderkoping	666	910	10, 998	1.37	6992
vadstena	581	778	7785	1.34	5613
valla	527	707	5583	1.34	–

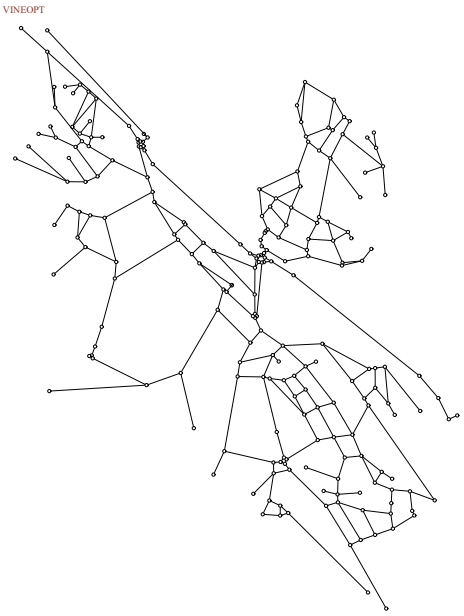

Fig. 6 Åtvidaberg (atvid), 224 nodes, 294 links

Fig. 7 Linghem, 836 nodes, 1129 links

left, the total time for each vehicle is shown. There one can identify the vehicles that takes the longest and the shortest time, and may consider moving links between them. In this example, one may consider moving links from vehicles 1 and 2 to vehicles 3 and/or 4.

In the middle green part, the best objective function value is shown, together with some more information, such as at which iteration the best value was found, and the lower bound.

In Fig. 2, we show how the network and allocation are brought into Vineopt. This allows single links to be changed in Vineopt. The resulting allocation can then be saved and used as starting solution in Snowplan.

In [16], we present preliminary tests, made to find interesting values of the parameters. (More parameters and values are tested than presented here.) One should remember that the method contains randomness, so all results should be regarded as approximate. In any case, in these tests, the best setting seems to be the following: MD 1, RS 1, SV 5, MV 2, IV 1, ChV 3, II 7, ST 1000, RC 0.3. In words: Mark passed streets done, revise allocation after this, start with k-means, put border links in the highest number set, move a link from a vehicle that does most to one that does least, try all combinations of moving cycles, swap the most and least used vehicles, try to move a cycle from an expensive tour to a cheap one, try once for each pair of vehicles. In simulated annealing,

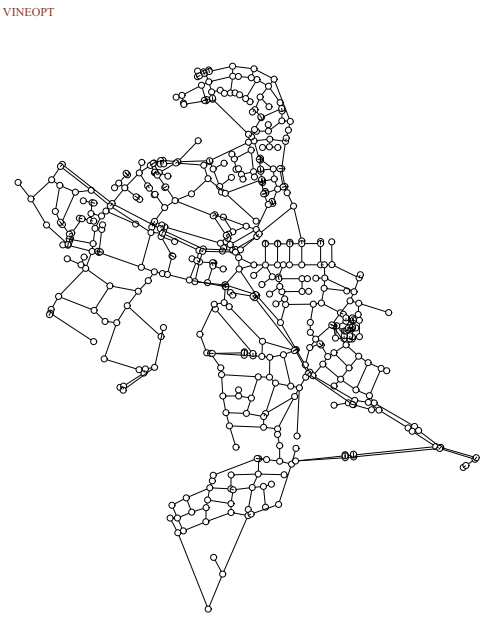

Fig. 8 Malmslätt (malmslatt), 489 nodes, 626 links

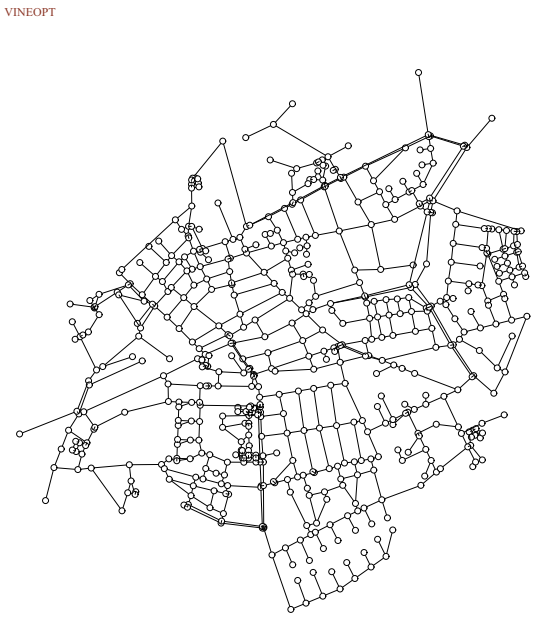

Fig. 9 Vadstena, 581 nodes, 778 links

do 7 interior iterations (with the same temperature), use start temperature 1000 and cooling factor 0.3. An observation is that these parameter settings make the method act more like greedy heuristics than metaheuristics, i.e., one searches for the best improvement in every step, and does not do random changes and leave the optimization to the metaheuristic. The reason is probably that each iteration is rather expensive, so we do not have time to do as many iterations as one would prefer in a metaheuristic.

We have chosen a number of interesting settings, i.e., variants of the method to try more. They are shown in Table 2.

Table 2 Settings to try

	MD	RS	SV	IV	ChV	II	RC
A	1	1	5	1	3	7	0.3
B	1	1	5	1	3	10	0.5
C	1	1	6	1	3	7	0.3
D	1	1	7	1	3	10	0.5
E	1	1	8	1	3	7	0.3
F	1	1	5	2	3	7	0.3
G	1	0	5	1	3	7	0.3
H	1	1	3	1	3	7	0.3
I	0	0	5	1	3	7	0.3
J	1	1	4	1	3	7	0.3
K	1	1	1	0	3	7	0.3
L	1	1	1	0	2	7	0.3

Additionally, MI = 30, MV = 2, ST = 1000 for all

We will also use the notation Ak, Bk, etc., which denotes setting A with k iterations and so on. For the setting A1, B1 etc., the method becomes a one-shot heuristic. If the starting solution is very good, the succeeding changes may not give much improvement, and the question is how many iterations one should spend on trying to improve the solution.

7.3 Tests of one-step methods

In [16], we give results for the settings with only one iteration, followed by one round of improvements. We use four vehicles, and find that the times are very reasonable.

We find that the settings K1 and L1 give bad solutions, while good solutions are obtained by several other settings. Some solutions are actually close to the lower bound, but most are not very close. Setting E1 gives the best solution for 8 instances, followed by G1 and C1, both giving the best for 5 instances. A conclusion is that one cannot be sure of good results using only one iteration.

7.4 Tests with 30 iterations

In several tables in [16] we present detailed results of our main tests with the chosen settings, A–L, using 30 iterations. In Table 3, we give the best and the worst of the results. Checking how many times each setting gave the best objective function value for $q = 4$ and 30 iterations, we get the following: A 8, G 6, B 4, C 4, E 4, F 4, D 2, H 1, and the others zero. The randomness makes it hard to pick one "best" setting, but here A seems to be a good choice, since it gives the best solution for eight problems.

Times are obviously longer here than with one iteration. However, it is still possible to use the method, from a practical perspective, as it is much faster to use this tool than to do it by hand.

Table 3 Results, best and worst objectives and times, settings A-L, $q = 4$

Name	Obj				Time				Lowb	
	Min		Max		Min		Max		l^0	l^1
atvid-s	B	3842	K	4881	K	1.00	J	1.73	3335	3690
brokind	D	5701	K	9134	K	3.02	J	5.12	4130	5225
colonia	G	989	K	1042	K	0.69	J	0.96	964	981
ekangen	G	4933	L	7265	K	2.30	I	3.92	4086	4707
mantorp-c	G	4473	L	7093	L	1.67	H	3.68	3604	4174
mantorp	A	7353	K	11477	K	3.36	C	5.75	5990	6948
rimforsa-o	B	3337	L	4704	K	1.28	F	1.93	2643	3105
skanninge-c	A	4375	L	6095	K	1.68	I	3.39	3665	4259
skanninge-n	E	2366	L	2840	L	0.88	G	1.38	2017	2248
skanninge	F	4450	K	6098	K	1.79	D	3.02	3703	4300
studentryd-e	F	1599	L	1732	L	1.03	H	3.08	1479	1559
sturefors	D	6418	L	9398	L	3.24	I	5.69	4904	5826
valla-liu	G	2657	K	3323	L	2.59	J	4.56	2279	2533
vikingstad	B	7234	K	10,646	K	3.21	B	5.69	5038	6341
atvid	G	10,443	K	13,963	K	5.40	B	8.92	8225	9657
borensberg	A	9451	K	12,660	K	4.34	E	8.54	7042	8785
kisa	C	8903	K	13,040	L	15.06	A	18.09	6476	7981
liu	F	5150	L	6353	J	13.81	H	19.08	4191	4931
rimforsa	B	5724	L	8837	K	6.18	J	14.07	4346	5266
ryd1	C	3993	L	4767	J	23.80	A	30.48	3550	3926
studentryd-int	C	2995	K	3739	K	7.15	D	11.16	2668	2926
studentryd	A	5242	L	6307	I	24.93	C	33.61	4450	5074
linghem	H	11,132	L	14804	I	171.83	K	274.90	9080	10562
ljungsbro	A	23,290	K	32700	I	154.99	K	229.68	18316	22207
malmslatt	E	10,411	L	14,130	I	41.85	K	50.67	8978	10187
mjolby	A	21,041	L	28,322	I	30.22	B	38.65	15668	18851
ryd-m	E	16,868	K	20,340	G	403.35	K	787.02	14437	16159
ryd	A	12,137	K	14,486	I	262.40	K	425.07	10678	11910
ryd2	E	13,948	K	16,957	G	286.28	K	522.63	12268	13707
soderkoping	A	17,566	K	22,537	G	95.05	K	131.28	14548	16913
vadstena	F	12,343	K	15,545	I	61.92	K	91.95	10532	12147
valla	G	9517	K	11,486	I	49.57	K	65.81	7779	9039

In Table 4, we investigate the setting A30 more thoroughly, by giving the results together with lower bounds and relative errors. The quality of the lower bounds may vary between instances.

7.5 Different number of iterations

In Table 5, we give the results for setting A with 1, 5, 30 and 100 iterations. The question here is the value of continued iterations. Since the methods contain randomness, we see cases where more iterations give a worse solution. That will obviously not happen within the same run.

Times grow almost linearly with the number of iterations. As there can be several iterations without improvement of the objective function value, improvement of the objective function value is not very regular. Usually, doing 100 iterations gives better solutions than doing 30 or less, so it is mainly a question of how much time one wants to spend.

7.6 Tests with more vehicles

We have also done test with more vehicles. In [16], we present tests with methods A30–L30 and $q = 5$. The number of best objective function values are the following: E 10, G 6, C 4, D 3, F 3, A 2, B 1, I 1, and the others zero. The worst objective function values were given by settings K and L. The minimal times, however, were given by K and L, and in a few cases G and I. The solution times for the first group is between 1

Table 4 Results with setting A30, $q = 4$

Name	Obj	Time	l^1	Err
atvid-s	3828.73	1.49	3690.50	0.036
brokind	5766.56	4.24	5225.82	0.094
colonia	1001.79	0.75	981.29	0.020
ekangen	5502.08	3.41	4707.38	0.144
mantorp-c	4612.99	2.36	4174.39	0.095
mantorp	7592.03	4.86	6948.47	0.085
rimforsa-o	3381.52	1.75	3105.75	0.082
skanninge-c	4623.21	2.49	4259.22	0.079
skanninge-n	2324.27	1.31	2248.61	0.033
skanninge	4738.71	2.60	4300.15	0.093
studentryd-e	1602.75	1.45	1559.96	0.027
sturefors	6443.14	4.74	5826.71	0.096
valla-liu	2764.75	4.01	2533.03	0.084
vikingstad	8445.48	4.72	6341.68	0.249
atvid	10, 753.67	8.41	9657.40	0.102
borensberg	10, 337.00	6.30	8785.55	0.150
kisa	8765.30	16.63	7981.47	0.089
liu	5335.31	15.18	4931.24	0.076
rimforsa	5893.20	8.07	5266.41	0.106
ryd1	4040.36	29.37	3926.69	0.028
studentryd-int	3062.97	8.91	2926.23	0.045
studentryd	5298.02	28.83	5074.35	0.042
linghem	11, 800.73	175.34	10, 562.68	0.105
ljungsbro	23, 566.40	159.42	22, 207.49	0.058
malmslatt	10, 450.04	44.52	10, 187.53	0.025
mjolby	21, 320.07	31.40	18, 851.91	0.116
ryd-m	17, 133.13	439.34	16, 159.83	0.057
ryd	12, 610.77	270.47	11, 910.95	0.055
ryd2	14, 768.43	295.82	13, 707.77	0.072
soderkoping	18, 538.84	99.72	16, 913.25	0.088
vadstena	12, 389.38	74.34	12, 147.58	0.020
valla	9529.93	58.79	9039.05	0.052

Err: Relative error to lower bound, (Obj-l^1)/Obj

Table 5 Results, objectives, with setting A with 1, 5, 30 and 100 iterations

Name	A1	A5	A30	A100
atvid-s	4292	4005	4014	3971
brokind	5761	5804	5916	5585
colonia	1021	994	1001	990
ekangen	5077	5542	5607	5334
mantorp-c	4511	4718	4557	4382
mantorp	7598	7458	7353	7544
rimforsa-o	3485	3387	3412	3340
skanninge-c	4492	4638	4375	4616
skanninge-n	2433	2383	2373	2414
skanninge	4712	4651	4470	4671
studentryd-e	1690	1651	1608	1603
sturefors	6477	6495	6988	6428
valla-liu	2718	2676	2692	2624
vikingstad	7180	7561	7574	7287
atvid	10381	10506	10444	10179
borensberg	11423	10062	9451	10065
kisa	9133	9393	9146	9496
liu	5161	5242	5236	5197
rimforsa	6298	5908	5875	5700
ryd1	4110	4008	4094	4082
studentryd-int	3083	2993	3102	3062
studentryd	5439	5412	5242	5275
linghem	11731	11226	11955	–
ljungsbro	23866	23525	23290	–
malmslatt	10867	10492	10658	–
mjolby	21177	21270	21041	–
ryd-m	17561	16912	16994	–
ryd	12612	12782	12137	–
ryd2	14359	14186	14876	–
soderkoping	18017	17935	17566	–
vadstena	12548	12438	12416	–
valla	11094	9648	9897	–

and 10 s, for the second group between 7 and 47 s and for the third group between 33 and 772 s. Picking a winner is not easy, but E and G could be candidates.

In Table 6, we present the tests with methods A30–J30 for six vehicles for the larger networks, as we believe that one would not use that many vehicles for smaller areas. Picking a winner here is even harder, but method I is quite fast, while the best results were found by C 3, A 2 and G 2.

7.7 Different numbers of vehicles

We have solved a couple of problems for different numbers of vehicles, from one up to eight, and then added a fixed cost for each vehicle. The results are reported in Table 7 for equal weights on time and cost, and 8 for weight on time twice the weight on cost, i.e., when it is more important to remove the snow quickly. We used settings A30.

The time for clearing obviously decreases when the number of vehicles increase, but the vehicle cost increases. One can then find a minimum of the total cost.

For equal weights on time and cost, we find that for Mantorp the total cost is minimized for four vehicles, while also six is good. For Åtvidaberg, four, five or eight vehicles seem to be best. For Vadstena, which is a somewhat larger city, the cost is minimal for eight vehicles. Again we recall that the methods use randomness, so one should not put too much

Table 6 Results, best and worst objectives and times, settings A-J, $q = 6$

Name	Obj		Time		Lowb
	Min	Max	Min	Max	l^1
linghem	11, 057	11, 705	299.70	317.15	10311
ljungsbro	22, 485	24, 238	273.34	303.09	1180
malmslatt	10, 296	11, 350	77.31	92.05	9961
mjolby	21, 290	23, 731	57.80	74.90	18048
ryd-m	16, 113	17, 061	667.29	744.05	15535
ryd	11, 935	12, 557	440.63	493.46	11570
ryd2	13, 697	14, 681	488.72	516.21	13247
soderkoping	16, 820	18, 603	170.62	202.87	16239
vadstena	12, 107	12, 534	128.81	140.17	11791
valla	9367	10, 084	98.98	109.23	8889

Table 7 Different numbers of vehicles, with $w^A = 1$ and $w^C = 1$

Veh	mantorp		atvid		vadstena	
	Obj	Time	Obj	Time	Obj	Time
1	10037	0.11	14372	0.20	18356	2.72
2	8161	2.22	11412	3.49	14203	43.08
3	7541	3.23	11026	4.98	12956	55.85
4	7233	4.77	10303	7.47	12475	74.48
5	7660	7.31	10404	13.42	12237	95.96
6	7364	11.61	10798	16.34	12306	139.74
7	7730	15.63	10567	25.07	12248	164.33
8	7812	20.14	10327	28.74	12130	211.49

Table 8 Different numbers of vehicles with $w^A = 2$ and $w^C = 1$

Veh	mantorp		atvid		vadstena	
	Obj	Time	Obj	Time	Obj	Time
1	14956	0.13	21458	0.31	27434	2.89
2	10513	3.74	14939	5.86	18792	44.77
3	9311	4.85	13137	7.57	16150	58.07
4	9104	7.59	12680	11.07	14837	74.25
5	8735	10.80	12825	14.85	14649	99.06
6	8375	12.65	11787	20.48	13859	130.47
7	8841	17.02	11931	26.31	13886	159.89
8	8874	19.02	12232	31.96	13763	201.42

trust in small differences. Perhaps the best conclusions of these tests are that one should not use less than three vehicles in Mantorp and not less then four in Åtvidaberg and Vadstena.

For a larger weight on time, we find that the minimum for Mantorp is found at six vehicles, for Åtvidaberg at six vehicles and for Vadstena at eight vehicles.

We see that the weights on cost versus time are important. Increasing the weight on time (i.e., that the snow is removed quickly) will increase the number of vehicles needed for the minimal total cost. Obviously, using this tool for a certain city, the authorities/users should put some effort in estimating the best values of these weights.

Estimating the fixed costs for vehicles may be rather difficult, since we are comparing investments for many years with daily cost of operations. We will look more at this question in the future. For now, the fixed costs are not much more than guesses, but we note that they give reasonable numbers of vehicles in our tests. The results in this paper are just an illustration of how our methods can be used to dimension the fleet for a certain area.

8 Conclusions and future work

The weighted k-rural postman problem appears when planning urban snow removal. We have developed heuristics for the problem, using metaheuristics, k-means clustering and local search improvements by moving cycles. The methods give fairly good solutions to real life problems. Solution times are reasonable and allows real life usage of the code. The graphical software gives possibilities for manual improvements.

By adding a fixed cost for each vehicle, we can compare the costs when using different numbers of vehicles.

Future research includes improved local search by moving cycles between non-adjacent tours, moving paths, and improved metaheuristics. We will also use the results of this problem for a more detailed routing for one vehicle at a time.

References

1. Ahr, D., Reinelt, G.: New heuristics and lower bounds for the min–max k-Chinese postman problem. In: Proceedings of the 10th Annual European Symposium on Algorithms, ESA '02, pp. 64–74. Springer, London (2002)
2. Ahr, D., Reinelt, G.: A tabu search algorithm for the min–max k-Chinese postman problem. Comput. Oper. Res. **33**(12), 3403–3422 (2006)
3. Benavent, E., Corberán, A., Plana, I., Sanchis, J.M.: Min–max k-vehicles windy rural postman problem. Networks **54**(4), 216–226 (2009)
4. Campbell, J.F., Langevin, A.: The snow disposal assignment problem. J. Oper. Res. Soc. **46**, 919–929 (1995)

5. Christofides, N., Campos, V., Corberán, A., Mota, E.: An algorithm for the rural postman problem on a directed graph. Math. Program. Study **26**, 155–166 (1986)
6. Cook, C., Schoenfeld, D.A., Wainwright, R.L.: Finding rural postman tours. In: Proceedings of the 1998 ACM Symposium on Applied Computing, pp. 318–326. ACM Press (1998)
7. Corberán, A., Prins, C.: Recent results on arc routing problems: an annotated bibliography. Networks **56**, 50–69 (2010)
8. Dror, M. (ed.): Arc Routing—Theory, Solutions and Applications (2000)
9. Edmonds, J., Johnson, E.L.: Matching, euler tours and the Chinese postman. Math. Program. **5**(1), 88–124 (1973)
10. Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, part I: the Chinese postman problem. Oper. Res. **43**, 231–242 (1995)
11. Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, part II: the rural postman problem. Oper. Res. **43**, 399–414 (1995)
12. Fu, L., Trudel, M., Kim, V.: Optimizing winter road maintenance operations under real-time information. Eur. J. Oper. Res. **196**, 332–341 (2009)
13. Ghiani, G., Laganá, D., Musmanno, R.: A constructive heuristic for the undirected rural postman problem. Comput. Oper. Res. **33**, 3450–3457 (2006)
14. Holmberg, K.: Heuristics for the rural postman problem. Comput. Oper. Res. **37**, 981–990 (2010)
15. Holmberg, K.: Urban snow removal: modeling and relaxations. Research Report LiTH-MAT-R–2014/08–SE, Department of Mathematics, Linköping University, Sweden (2014)
16. Holmberg, K.: Heuristics for the weighted *k*-Chinese/rural postman problem with a hint of fixed costs with applications to urban snow removal. Research Report LiTH-MAT-R–2015/13–SE, Department of Mathematics, Linköping University, Sweden (2015)
17. Holmberg, K.: Map matching by optimization. Research Report LiTH-MAT-R–2015/01–SE, Department of Mathematics, Linköping University, Sweden (2015)
18. Holmberg, K.: On using OpenStreetMap and GPS for optimization. Research Report LiTH-MAT-R–2015/15–SE, Department of Mathematics, Linköping University, Sweden (2015)
19. Holmberg, K.: The (over) zealous snow remover problem. Research Report LiTH-MAT-R–2016/04–SE, Department of Mathematics, Linköping University, Sweden (2016)
20. Islam, S.: Application framework for snow removal routing problem. Master's thesis, Linköping University (2010). LIU-IDA/LITH-EX-A–10/012–SE
21. Labelle, A., Langevin, A., Campbell, J.: Sector design for snow removal and disposal in urban areas. Socio-Econ. Plan. Sci. **36**, 183–202 (2002)
22. Lloyd, S.: Least squares quantization in PCM. Inf. Theory IEEE Trans. **28**(2), 129–137 (1982)
23. Osterhues, A., Mariak, F.: On variants of the *k*-Chinese postman problem. Diskussionsbeiträge des Fachgebiets Operations Research und Wirtschaftsinformatik 30, Universität Dortmund (2005)
24. Pearn, W.L.: Solvable cases of the *k*-person Chinese postman problem. Oper. Res. Lett. **16**, 241–244 (1994)
25. Perrier, N., Langevin, A., Amaya, C.A.: Vehicle routing for urban snow plowing operations. Transport. Sci. **42**, 44–56 (2008)
26. Perrier, N., Langevin, A., Campbell, J.F.: A survey of models and algorithms for winter road maintenance: Part I: system design for spreading and plowing. Comput. Oper. Res. **33**, 209–238 (2006)
27. Perrier, N., Langevin, A., Campbell, J.F.: A survey of models and algorithms for winter road maintenance: Part II: system design for snow disposal. Comput. Oper. Res. **33**, 239–262 (2006)
28. Perrier, N., Langevin, A., Campbell, J.F.: A survey of models and algorithms for winter road maintenance: Part III: vehicle routing and depot location for spreading. Comput. Oper. Res. **34**, 211–257 (2007)
29. Perrier, N., Langevin, A., Campbell, J.F.: A survey of models and algorithms for winter road maintenance: Part IV: vehicle routing and fleet sizing for plowing and snow disposal. Comput. Oper. Res. **34**, 258–294 (2007)
30. Razmara, G.: Snow removal routing problems - theory and applications. PhD dissertation, Linköping University, Sweden. Linköping Studies in Science and Technology. Dissertation no. 888 (2004)
31. Trudel, M.R.: A mathematical model for winter maintenance operations management. Master's thesis, University of Waterloo (2005)