Department of Mathematics

The (Over) Zealous Snow Remover Problem

Kaj Holmberg

LiTH-MAT-R--2016/04--SE

Department of Mathematics
Linköping University
S-581 83 Linköping

The (Over) Zealous Snow Remover Problem

Kaj Holmberg

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

April 12, 2016

Abstract: Planning snow removal is a difficult, infrequently occurring optimization problem, concerning complicated routing of vehicles. Clearing a street includes several different activities, and the tours must be allowed to contain subtours. The streets are classified into different types, each type requiring different activities. We address the problem facing a single vehicle, including details such as precedence requirements and turning penalties. We describe a solution approach based on a reformulation to an asymmetric traveling salesman problem in an extended graph, plus a heuristic for finding feasible solutions. The method have been implemented and tested on real life examples, and the solution times are short enough to allow online usage. We compare two different principles for the number of sweeps on a normal street, encountered in discussions with snow removal contractors. A principle using a first sweep in the middle of the street around the block, in order to quickly allow usage of the streets, is found to yield interesting theoretical and practical difficulties.

1 Introduction

Snow removal is an important problem in northern countries. It seems that changes in the weather are becoming more significant, and the yearly amounts of snow in Sweden changes very much. This means that last year's plan may not be suitable for this year. The need for making new plans seems to increase, which in turn gives a need for tools that help making good plans for snow removal.

In this paper we study the snow removal process in urban areas (with focus on the circumstances in Sweden, which we however believe is not very different from other countries). In Perrier, Langevin, and Campbell (2007) and Perrier, Langevin, and Amaya (2008), the Canadian and North American situation is described.

A city is divided into areas. For each area, a contractor is hired to take care of the snow removal. There is no point in optimizing the snow removal for more than one area at a

time, since the contractors do not cooperate. The street network in such an area is not huge.

We will study snow removal after a snowfall. A street that has been cleared of snow at one time is free of snow (for the rest of the time considered). In the paper Holmberg (2014), we give a detailed (and very complex) model for the complete snow removal problem. For practical purposes, this model is too difficult to be solved exactly. The problem of allocation of streets to identical vehicles is treated in Holmberg (2015a). This may be a part of a multilevel decision process for producing good snow removal plans.

In this paper, we will study the optimization problem facing a single snow remover (i.e. the driver of a snow removing vehicle). We assume that it has been decided which subarea the driver should cover, i.e. which links that shall be treated by the vehicle. Other vehicles (drivers) will take care of other subareas, and we don't consider any interaction between the subareas/drivers. (This is more or less how it is done today in practice.)

The task is simply to plan the tour for the vehicle. In its most simplified form, it will be a Chinese postman problem. However, there might be both one-way and two-way streets, i.e. both directed and undirected links. Furthermore, there may exist links that don't need to be treated, but can be used for transportation of the vehicle.

A detail that has occurred in our discussions with Swedish contractors seems to be quite different from what is usually described in the literature, for example in Perrier et al. (2008). That is that a normal city street is sometimes cleared *three* times, first in the middle, then on one side, and finally on the other. The two last sweeps must be done in the correct direction (on the right hand side of the street) but the first sweep can be done in any direction. This is a significant complication compared to the case when only one sweep in each direction is required. If that was the case, a street could be split into two directed arcs, each requiring one sweep.

The complication is not only that the number of sweeps is higher, but also that there are both directed and undirected sweeps. One may compare to the Chinese postman problem, where the directed (and the undirected) case is polynomially solvable, but the mixed case is *NP*-hard.

The middle sweep is done to enable traffic as soon as possible. Often the middle sweep is done around the block, before doing the sides, in order to make the streets usable. Therefore we don't want a plan that does all three sweeps on a street in direct succession before going on to the next street.

As in Holmberg (2014), we will break down operations into unsplittable tasks. Examples of such tasks are the following.

- Clear the middle of a street.
- Clear the right side of a street.
- Clear the left side of a street.

- Clear the middle of a street a second time. (Only for wider streets.)
- Clear a turning space.
- Additional clearing at crossings.

There exists other tasks, such as clearing bicycle paths, pedestrian paths, bus stops etc, but they are usually done by other vehicles, and are not considered here.

An initial idea could be to group together the three sweeps of a street into one task, by always clearing the middle, left side and right side of a street in direct succession. However, this way we lose one of the main advantages of the middle sweep, so we will not do this here.

There are precedence constraints for some tasks. The middle of a road must be cleared before the left and right sides are cleared, and the streets leading to a crossing must be cleared before the final clearing of the crossing.

Most tasks have defined starting and ending points. Sometimes they are the same, for example for a turning space, and sometimes they are at opposite ends of a street.

Clearing the middle of a street can be done in any direction. This can be handled by defining this as two different tasks, one in each direction. Then these two tasks are grouped together, and we only require that one task in each group is done.

There are several possible objectives for this problem. One is to minimize the time until everything is ready. However, it is also interesting to minimize the transportation distance (while not clearing). The contractor that does the clearing might want to minimize the total cost for the operation. From an environmental point of view, one might want to minimize the pollution caused by the vehicle. However, many of these objectives are more or less the same, especially when only considering one vehicle. Minimizing the distance traveling while not doing anything useful will probably also in principle minimize the pollution, and it should also lead to a minimal finishing time. Therefore we will here use the first one, minimizing the final end time.

There might be a certain time needed for switching between two tasks, for example time for turning the vehicle around at the end of a street. The issue with turning penalties has been treated in for example Soler, Martinez, and Mico (2008).

We need the following indata. We need to know the time (or cost) for each task, and the time (or cost) for each possible transportation link. We need to know start and end point of each task, and the time needed to switch between each pair of tasks/links, which includes times for turning.

The street network and all task characteristics can be found in different data bases, such as OpenStreetMap. See Holmberg (2015b) for details.

The times for tasks are known very approximately, based on an average speed for the vehicle. However, the operators will from now on be required to equip each vehicle with a GPS unit, which gives data from which we may extract times for all tasks. Task times

will therefore be available with better and better accuracy. (Analyzing the GPS tracks will require map matching, which is treated in Holmberg (2015a).)

The links are categorized according to the number of sweeps that are required to clear the street completely. In Holmberg (2015b), we give 8 link types, of which however the first three don't admit cars, and some make a difference whether or not they admit lorries. There is also a type for links not needing snow removal (i.e. when someone else takes care of that). Here we simplify it a bit by using only the following types.

- 1: Narrow street (needing only one sweep).
- 2: Small ordinary road (2 sweeps).
- 3: Normal ordinary road (3 sweeps).
- 4: Larger ordinary road (4 sweeps).

In Holmberg (2015b) we give details about how these link types are extracted from OpenStreetMap data.

From each link, we use its type to create a number of tasks. For a normal ordinary road there could be four tasks, out of which two form one group which corresponds to the undirected middle sweep. For narrow streets, the link task often is undirected, which gives a group with two tasks. Usually there is one task per node, which depends on the degree of the node.

Here we make a distinction between the (over) zealous snow remover, who uses three sweeps for ordinary roads (type 3), and other snow removers, who use only two sweeps on this type of road. We will study the effect of this difference, and will use the notation ZSRP for the (over) zealous snow remover problem, and SRP for the (more normal) snow remover problem.

2 Structures

In the most aggregated form, all the tasks connected with a certain street segment is aggregated into one task. This means that all the tasks (middle, left side, right side, crossing/turning point etc) are assumed to be done in sequence, and the time for this aggregated task is the sum of times for the detailed tasks. In order to simplify further, we assume that each task starts at one end of the street, and ends at the other.

If all streets in the network need to be treated, and they all are undirected, we have the standard Chinese postman problem. This problem is easily solvable to optimality with known methods. In principle one duplicates edges so that all vertices get even degree. If this is done at minimal cost, any Euler cycle in the resulting graph is an optimal tour.

If all street tasks are directed, we have a directed Chinese postman problem, which also is solvable with known methods. This can be the case when there are two tasks for each street, one in each direction.

However, if some streets are directed (one-way) and others are not, we have the more difficult mixed Chinese postman problem. One way of dealing with undirected tasks is to define a group with two anti-parallel directed tasks, and require one task in each group to be done. We then get the generalized Postman Problem, which is NP-hard.

If not all streets in the network need to be treated, we have the rural postman problem. As before, we can have undirected, directed and mixed versions of this problem. All of them are NP-hard, but for the undirected rural postman, there are efficient heuristics.

Going further into the details, there could be a disadvantage in having to turn a vehicle around on a small street. It might be more efficient to do one task for several streets, for example around the block, then turn around and do another task around the block again, then to do both tasks on each street before going on to the next one.

This gives a need for turning penalties. There are possibilities of extending the networks for handling turning penalties. They require directed arcs, and make the problem harder to solve.

There could also be precedence constraints between the different tasks. There are especially two cases which occur in our examples. One is when a street requires three runs of the vehicle, one in the middle, one on the left side and one on the right side. Then the middle run should be done first, before the other two. (In addition, the middle run may be done in any direction, even if the two side runs are directed.)

Another case concerns crossings and turning areas on dead end streets. Then all adjacent streets must be cleared first, and then additional work is made in the crossings and turning areas. More precisely, all tasks leading into a node must be done before the node task is done, while we allow outgoing tasks after the node task.

One might argue that the additional work at crossings should be done the last time the vehicle passes the crossing. However, before the plan is made, we don't know which task this will be.

3 Mathematical notation

In Holmberg (2014), we formulate a mathematical model for the detailed snow remover's problem. The model is based on the street network and discretization of time. We will not repeat the model here, but in principle one may use it, by simply removing the index for vehicles. Also some constraints dealing with the interaction between different vehicles will be removed. However, this model will not give an efficient way of solving the problem.

Let m denote the number of links in the network, n the number nodes, p the number of tasks, and g the number of groups.

We assume that the snow remover can do all the given tasks and that the vehicle can use all the given links. All other tasks and links are removed from the problem. What is left to decide is the order of the tasks and the tour for the vehicle.

Let us just define notation for the indata.

d_j^T : time needed for the vehicle to do task j .

d_l^L : time needed for transporting the vehicle along link l .

D : depot for the vehicle, i.e. the node where it starts and must return to.

s_j^T : starting point for task j .

e_j^T : ending point for task j .

s_l^L : starting point for link l .

e_l^L : ending point for link l .

G_g : the set of tasks that belong to group g .

g_j is the group that task j belongs to. ($G_g = \{j \in T : g_j = g\}$)

(Often one task is one group.)

$\rho_{gj}^{GT} = 1$ if group g must precede task j .

$\rho_{gl}^{GL} = 1$ if group g must precede usage of link l .

$d_{ll'}^{SFLFL}$: time needed for the vehicle to switch directly from using link l forwards to using link l' forwards.

Likewise we define $d_{ll'}^{SFLBL}$, $d_{ll'}^{SBLFL}$ and $d_{ll'}^{SBLBL}$ for the time needed for the vehicle to switch directly between the links when one or two of them are used backwards. Switching times between tasks are calculated by identifying which links the tasks are done on.

4 Creation of tasks

Given the street network and the link types, we create the tasks that need to be done.

First we remove all links that cannot be used by the vehicle in question. Optionally we may also remove parts of the network that most certainly will not be used by the vehicle, i.e. parts that lie far away from the depot and the links to be cleared.

Then we step through all links and create the tasks and groups, depending on the link type. Each task gets a starting node, s_j^T , and an ending node, e_j^T , the time needed for doing it, and a group. If there are two tasks in the group, we also save the index of the other task in the group, since this is useful in the calculations. We also note the presence of predecessors and/or successors, as well as on which link the task takes place.

The number of link tasks is up to 6 for each link.

After this, we step through all nodes and construct the node tasks. There is one node task per node, but the time needed depends on the type of node (crossing, turning space). Also here we note all predecessors.

When all tasks are defined, we fill in the switching times between all pairs of tasks, and sum up all predecessors and successors to each task.

The exact total number of tasks is not known before we have gone through all links and checked their type. p will lie between $2m + n$ and $6m + n$. (We assume that the most narrow links, needing only one sweep, are undirected, and thus will need a group.)

5 Solution methods

5.1 Lower bound

In Holmberg (2014), several relaxations of the complete snow removal problem are described and tested. Here we only have one vehicle, and use the simple lower bound of summing up the total work that needs to be done. In other words, we ignore the need for a tour, i.e. omit all the transportation of the vehicle.

5.2 Finding a feasible solution

If the order of the tasks is given, we can find a correct and feasible solution to the problem as described below. (This will be a simplified version of a procedure we have used for several vehicles.) What remains to determine is the exact times at which the tasks will start and the movement of the vehicle between tasks (when they are not adjacent).

Let us here assume that the order is correct, i.e. that it is feasible to do the tasks in that order, without violating the precedence constraints. (We will return to the question of ensuring this later.)

We build up the tour by increasing the time t step by step. Let the order be given by o_k giving the index of the k -th task to be done.

Consider a point in time, $\hat{t}$, where the vehicle is positioned at a location, denoted by π , and the tasks $o_1 - o_{k-1}$ are done. (In the beginning, it is located at the depot, $\pi = D$, and no task is done.)

The next task to be done is then o_k . If $s_{o_k}^T = \pi$, the vehicle is in the correct node for starting the task. Otherwise it has to move to that position. We then need to find the shortest path through the network from π to $s_{o_k}^T$.

This is enabled by initially finding the shortest path between all pairs of nodes with the method of Floyd-Warshall. Then we only need to extract the path by a recursive usage of the predecessor indices. (However, these distances will not contain the switching times.)

We then let the vehicle travel along these links. In each step, we keep account of the time. When a link is traversed and a task is made, we advance the time according to what is needed by the link or task. Here the switching times are taken into account, since we know between which links the vehicle travels.

The vehicle is marked busy for the following d_j^T time periods after starting task j , and for d_l^L time periods after starting transportation on link j .

The tour is started and ended at the depot.

Let us now give an algorithm for this. Let t^V denote the “freeing time” for the vehicle, i.e. when it will be free for new tasks.

As preprocessing we have calculated the shortest distance between any two nodes i and j . This also gives p_{ij} which is -1 if the shortest path between nodes i and j is the direct link from i to j , and equal to k , if the shortest path from i to j goes from node i to node k (and on towards node j).

1. Let $t = 1$ and $\pi = D$. Let $t^V = 1$. Let $k = 1$.
2. The next task to do is o_k . Can the vehicle do task o_k at the present time? It is possible if $\pi = s_j^T$, all precedences for task o_k are made, and task/travel/switching times from the previous task/link have passed. If so, go to step 7.
3. If the task/travel/switching times from the previous task/link have not passed, advance the time counter sufficiently, and go to 2.
4. If all precedences for task o_k are not made, the order given does not obey the precedence requirements. Quit the procedure, and change the order.
5. If $\pi \neq s_j^T$, the vehicle must travel from node π to node s_j^T . Use the predecessor index for the shortest path, $p(\pi, s_j^T)$, to identify the next link in the path, l .
6. Use link l :
 - (a) If forwards (i.e. $\pi = s_l^L$): Set $t^V = t + d_l^L$. Set $\pi = e_l^L$.
 - (b) If backwards (i.e. $\pi = e_l^L$): Set $t^V = t + d_l^L$. Set $\pi = s_l^L$.

Go to 2.
7. Do the task o_k : Set $t^V = t + d_{o_k}^T$. Set $\pi = e_{o_k}^T$. Set $k = k + 1$. Go to 2.

5.3 Asymmetric traveling salesman problem formulation

A lot of research has been done on the traveling salesman problem, and efficient codes exists, so a natural question is if that work can be used for our problem. For this purpose, we formulate the following ATSP (asymmetric traveling salesman problem). First create a “task graph” by introducing two nodes for each task, and a directed arc from the starting node to the ending node, see figure 1.

Figure 1: Task arc.

Then we connect the end nodes of the “task arcs” to the start nodes by adding arcs representing the minimum cost path from each end node to each start node. Especially, when the two “task nodes” represent the same node in the street network, the arc cost is zero. We avoid adding arcs that may not be used because of precedence requirements.

This gives a directed graph where each start node of a task arc only has one leaving arc, the task arc itself, and each end node only has one incoming arc. Since each task must be done once, this means that each node must be passed exactly one, thus we have an ATSP, see figure 2.

Figure 2: Task graph with three tasks for ATSP.

If task 1 must precede task 2, we do *not* include arc $(T2b, T1a)$. Turning penalties are easily handled by this approach by simply adding this cost to the relevant arcs.

A solution to this ATSP gives an order in which the tasks should be made. However, even if direct arcs violating the precedence requirements are removed, the solution is not certain to obey the precedences. If the order for a vehicle breaks the precedence constraint, we may reorder the problematic tasks.

Tasks with precedence relations are adjacent. Assume that task 1 must precede task 2. Since one can't go directly from 2 to 1, the tour must visit a third task (3) between them in order to violate this precedence. Task 3 will not be adjacent to both task 1 and 2, so there will be an additional cost to do so. Often that tour will not be cheapest, and thus will not be obtained. This reasoning makes violated precedences unlikely for link tasks within the tour for one vehicle.

Unfortunately this is not true for node tasks, since it all takes place at the same node. Therefore, a node task may well occur before the corresponding link task.

The distances used in the ATSP are based on shortest path distances in the street network, which may not exactly reflect the final solution time.

A so far unanswered question is how to handle the undirected tasks, modeled by the groups. This yields a case of the *generalized* ATSP, where one node from each *set* must be visited.

Luckily there exists a reformulation from a generalized ATSP to a normal ATSP, Noon and Bean (1993). One adds zero cost arcs in a cycle between the nodes in a set, and switches the outgoing arcs one step along the cycle. The thought is that one enters a node, i , in the group, takes the free (artificial) tour in the cycle within the group, and leaves at the node just preceding node i in the group cycle. This is interpreted as entering and leaving the group at node i , and not visiting any other node in the group.

For our special graph, we note the following. All sets include only (at most) two nodes. Consider tasks 1 and 2, of which only one needs to be done. We do not include the arcs $(T2b, T1a)$ and $(T1b, T2a)$, since these arcs can only be used if we plan to do both

tasks. T1a and T2a constitutes one set, and T1b and T2b the other, so we add arcs connecting these pairs of nodes.

In addition the outgoing arcs from T1b and T2b are exchanged. The resulting tour leaves a group at “the other” node. This yields a good way of handling the difficulty with groups.

In the figure above, entering node T1a, going to T2a, then traversing to T1b, and then leaving via node T2b, means doing task T1.

It is unlikely that both (T1a, T2b) and (T2a, T1b) are used, since it costs nothing to go between T1a and T2a and between T1b and T2b. However, in degenerate cases this may occur. Therefore we may add a constant to all such task arcs, so that they don’t get used unnecessarily.

In order to enforce a visit to the depot, we include a separate node representing the depot. From this node, we include arcs to all starting node that may come first in the tour, i.e. those that have no predecessors. Likewise, we include arcs to the depot node from all ending nodes that may come last in the tour, i.e. those that have no successors. The costs on these arcs are the shortest distance between the nodes (which is equal to zero if they are the same node).

The increase in graph size due to the depot is moderate. The number of nodes is only increased by one, but the number of arcs is increased. To solve the resulting ATSP, we can use a heuristic like LKH, Helsgaun (2000).

Occasionally the tour received was the wrong way, so in these cases the tour was reversed. We also translate the tour so that it starts at the depot.

Then the tour was followed and the tasks were identified. It can be noted that all task arcs go from left to right, while all transportation arcs go from right to left. The group construction described above disturbed the clean structure somewhat, but it is still easy to identify in which order the tasks were passed.

We also keep the interpretation of the tour in the original nodes, and add the additional nodes passed in transportation.

It turned out that node tasks often were misplaced with regards to the precedence requirements. The ATSP-tour may place a node task at any time the correct node was passed, while the correct placement would be at the last time the node was passed.

Furthermore, the choice of which task in a group to make was sometimes incorrect with regards to the adjacent node task. A node task can be made after the vehicle has cleared that last link leading *into* the node, while it is allowed to clear links leaving the node

after the node task is made.

Therefore we found it best to recreate the task order, while following the proper tour (in the original nodes) around. When doing this, the first task encountered in a group was done, which corrected these errors. Furthermore, the node tasks were inserted in proper order, at the last passage of the node.

A variation of the method is to skip the node tasks when creating the ATSP graph. This gives a tour without the node tasks, but with all other tasks. The node tasks are then inserted at the proper place, when doing the re-tasking.

There still remains an occasional problem. For a normal road (type 3) the ZSRP tour passes a link three times, not all in the same direction. We must then let the first pass be the middle sweep, and if the two later passes are in different direction, this works. However, if the first sweep is in one direction, and the following two is in the other, this will fail. The change would require that one of the later sweeps change direction, which will change the tour.

We however found it best to insert the last task in the proper direction as early as possible, with respect to precedences. We then afterwards need to use transportation back to where the tour continues. This makes the solution slightly worse, but is the only simple way we can make sure that the precedence requirements are obeyed.

The size of the graph is the following. We have two nodes for each task plus one for the depot, so we get $2p + 1$ nodes. There are p task arcs, and we will add 4 arcs for each group with more than one task, of which there is not more than $2m$. Then we add up to $2p$ arcs to and from the depot (in practice much less). Finally we add at most an arc from each node in the second level to each node in the first level, which yields p^2 arcs. We recall that p is almost $6m + n$. In complexity terms, p is $O(n^2)$, so the number of nodes in the ATSP-graph is $O(n^3)$. Summing up, the growth in problem size is polynomial, and the graph is not complete.

6 A small numerical example

Let us now give a small example for illustration. The network consists of two links, one from node 1 to node 2 with type 3 and length 236 m, and one from node 1 to node 3 with type 2 and length 275 m. The depot is at node 1. Furthermore, we have turning place time 4 and crossing time 2, while the task time is $1.0 + 0.006 * t$, and the transportation time is $1.0 + 0.005 * t$, where t is the length of the link. Time for turning is 1 for a u-turn and 0 otherwise.

In the ZSRP, link (1, 2) gives rise to four tasks (T1, T2, T3 and T4), two in each direction, where the first two (T1 and T2) represent the middle sweep and form one group, while the other two (T3 and T4) give one group each. Link (1, 3) gives rise to two tasks (T5 and T6), one in each direction, in different groups.

Finally there are three node tasks (T7, T8 and T9), one at each node, where the tasks at

Figure 3: Graph for example: Task arcs and depot arcs.

nodes 2 and 3 (T8 and T9) are turning places while T7 at node 1 represents a crossing.

This yields a total of 9 tasks and 8 groups. Precedences are the following. The middle sweep, group 1 (tasks T1 or T2), must come before tasks T3 and T4. All incoming tasks to a node must come before the node task, so tasks T1 and T3 must come before task T8, tasks T2, T4 and T6 must come before task T7, and task T6 must come before task T9.

Constructing the directed graph for the ATSP, we first add the 18 start and end nodes for each task, plus one node for the depot, and insert the task arcs $(1, 2), (3, 4), \dots, (17, 18)$. We thus have a starting node with odd number and an ending node with even number for each task. The costs on these arcs are equal to the time needed for doing the tasks. We note which “real” node each ATSP node corresponds to. Here, nodes 1, 4, 5, 8, 9, 12, 13, 14 and 19 correspond to real node 1, etc.

Then we add arcs from each task ending node that may be last in the tour, i.e. tasks that have no successors, which in this case are the nodes 14, 16 and 18, to the depot node. We also add arcs from the depot node to each task starting node that may be first in the tour, i.e. tasks that have no predecessors, which in this case are the nodes 1, 3, 9 and 11. The costs on these arcs are set to the distances,

After this step we have 16 arcs. Figure 3 show the graph after this step.

Now we add arcs connecting the tasks. First we add arcs with cost zero between each task ending node and task starting node that correspond to the same real node. Figure 4 show the graph after this step.

Then we add arcs from task ending nodes to task starting nodes with costs equal to the distance between the corresponding real nodes. (We don’t have a picture of this graph,

Figure 4: Graph for example: Real node identity arcs.

as it becomes quite cluttered.)

We do not add arcs from the ending node of a task to the starting node of a second task, if the second task must precede the first. Likewise, we do not add arcs between tasks in the same group, since they will never be both done. One may also refrain from adding a link between two tasks, if we know that another task must be done between them. We have however not implemented this test.

Disregarding the depot node, the graph can be drawn as a bipartite graph, with all task arcs going from left to right, and all transportation arcs (including the ones representing the same real node) going from right to left. At this stage, there are no arcs within the two node sets. There is only one outgoing arc from each node in the left set, and only one incoming arc to each node in the right set, and these are the task arcs.

(If we wish, we may double the depot node, and use a task arc for this too, so that it can be inserted into the same bipartite structure.)

After this, we modify the graph with respect to the groups. In this example, only tasks T1 and T2 form a group. We therefore add zero cost arcs in both directions between nodes 1 and 3, and between 2 and 4. Then we shift the task arcs, so that task arc T1 goes from node 3 (instead of 1) to node 2, and task arc T2 goes from node 1 (instead of 3) to node 4. Finally we swap all outgoing arcs between nodes 2 and 4. This is shown in figure 5.

In this graph, doing task T1 will mean that the tour enters node 1 (the “correct” one), but then goes to node 3, then to node 2 and then to node 4, and continues from there (the “wrong” node).

In this example, this gives a total of 80 arcs. The corresponding ATSP is solved by

Figure 5: Graph for example: Group.

LKH in much less than 0.1 seconds (which is zero in the output of the code).

The tour received from LKH is 1 3 19 14 13 12 11 18 17 10 9 8 7 16 15 6 5 2 4. However, the direction is wrong, so we turn the tour around. Furthermore we shift the tour so that it start with the depot node, 19. We then get the tour 19 3 1 4 2 5 6 15 16 7 8 9 10 17 18 11 12 13 14. It is shown in figure 6. The dashed line between node 2 and 5 denotes a transportation between two different nodes.

Then we check the precedence requirements. In some cases, they are satisfied, but usually not. In that case, we follow the tour in the real nodes, and insert the tasks as early as possible. Often this means shifting node tasks to the last visit of the node, and occasionally changing which task in a group is done.

After this, we have a correct order, and end by using the procedure for finding a feasible solution, complete with exact times for each task etc. In this example, the solution is as follows. Do task 1 (1 - 2) at time 1 - 3, do task 3 (2 - 1) at time 3 - 5, do task 2 (1 - 2) at time 5 - 7, do task 7 (2 - 2) at time 7 - 11, use link 1 backwards (2 - 1) at time 11 - 13, do task 4 (1 - 3) at time 13 - 15, do task 8 (3 - 3) at time 15 - 17, do task 5 (3 - 1) at time 17 - 19, do task 6 (1 - 1) at time 19 - 21.

This yields a total cost (time) of 21. The lower bound calculated is 18. (However, we believe that the solution is optimal.) The solution time is 0.003 seconds.

Doing the same with the SRP, the first group is not needed. We then get 7 tasks and no group (with more than one task). We have 15 nodes, and the task arcs and depot arcs now sum up to 14. After adding all connection arcs, we get 52 arcs. The optimal objective function value now is 17, and the solution time is 0.002 seconds.

Figure 6: Graph for example: Tour.

7 Computational tests and evaluation

7.1 Implementation

The methods were implemented in C, and the tests were run on an Acer Aspire X3 X3995 3.4GHz, running Linux, Fedora Core 17. The machine has four CPUs, but only one was used in the test runs. The asymmetric traveling salesman problems were solved with the excellent heuristic LKH, Helsgaun (2000), and data communication was made via files.

7.2 Test problems

The method was tested on two groups of problems, one with artificial, rather small networks, called *mini*, and one with small cities and parts of cities obtained from OpenStreetMap. Details of the extraction process can be found in Holmberg (2015b). A certain filter was applied to the OSM-data, leaving only streets (highways) suitable for cars, but not items labeled “cycleway”, “pedestrian” and “footway”. Some other preprocessing was done, for example eliminating nodes with degree two by simply adding the two adjacent links into one. Therefore the curvature of the link is not visible, but the distance is correct.

Furthermore the cities were divided into parts, suitable for one vehicle. For this, a procedure as the one described in Holmberg (2015a) can be used.

Finally some nodes with degree one were eliminated, as these parts are uninteresting for

optimization, since there is in principle only one way of treating them. In the future, we will develop a procedure that automatically removes all subgraphs that are trees, makes a plan for them separately, and adds these parts afterwards. Presently, only the most obvious parts of the graphs were removed, but many nodes with degree one still remain. We have removed some instances (very small cities), since they in essence only were trees.

Problem set 1 is used in order to enable detailed analysis of the model and solutions, while problem set 2 obviously contains more realistic instances, and are considered as our main tests.

In the real networks, we often found that parts of the cities were almost separable, with only one or very few links connecting the parts. In such a case the routing can be done separately in the different parts. Either the parts are done by different vehicles, or the tour for one vehicle is obtained by first doing the tour for one part and then adding the tour for other part. This will not make the solution much worse, and it is not efficient to optimize in unnecessarily large graphs.

In figure 7, we show the network atvid, and in figure 8, we show how the network was divided into three separate parts, where the routing can be done separately. As can be seen, the separation is very natural, with very few links in common.

Figure 7: Åtvidaberg.

Figure 8: Åtvidaberg separated.

In figures 9 - 12, we give pictures of some of the mini networks, and in figures 13 - 20, we give pictures of some of the city networks.

Figure 9: Example: mini-03, 5 nodes, 5 links

Figure 10: Example: mini-06, 9 nodes, 11 links

Figure 11: Example: mini-08, 9 nodes, 15 links

Figure 12: Example: mini-09, 18 nodes, 26 links

VINEOPT

Figure 13: Example: askeby, 53 nodes, 56 links

VINEOPT

Figure 14: Example: borensberg-b, 103 nodes, 131 links

VINEOPT

Figure 15: Example: boxholm-1, 90 nodes, 119 links

VINEOPT

Figure 16: Example: brokind-1, 89 nodes, 101 links

VINEOPT

Figure 17: Example: mantorp-b, 68 nodes, 83 links

Figure 18: Example: skanninge-a-1, 68 nodes, 86 links

Figure 19: Example: valla, 69 nodes, 85 links

Figure 20: Example: vikingstad-a-2, 63 nodes, 86 links

7.3 Computational results

We have done computational tests to see if the realistic instances can be solved in reasonable time. We also wish to compare the ZSRP and SRP, both when it comes to difficulty (solution time) and cost (objective function value).

The computational tests on problem set 1, “mini”, are presented in table 1 for the ZSRP and in table 3 for the SRP. The tests on problem set 2, the city parts, are presented in table 2 for the ZSRP and in table 4 for the SRP.

In the tables we give m , the number of nodes, n , the number of links, p , the number of different tasks, pg , the number of groups, low , the lower bound, an , the number of nodes in the resulting asymmetric traveling salesman problem, aa , the number of arcs in the resulting asymmetric traveling salesman problem, $atim$, the time needed for solving the asymmetric traveling salesman problem as reported by LKH, obj , the objective function value of the final feasible solutions, and $time$, the total solution time (including $atim$).

Our first conclusion is that the mini-cases are solved very quickly, in fractions of a second. The larger instances take longer time, and the time is completely dominated by the time to solve the ATSP. On the other hand, all instances are solved in less than 30 seconds, so it is perfectly possible in practice to solve them each day, before starting the work.

Comparing SRP and ZSRP in tables 2 and 4, we find that both solution times and objective function values are decreased in SRP. Taking the first problem, askeby, as an example, the solution time was decreased from 3.53 seconds to 0.73 seconds. The objective function value, i.e. the snow removal time, was decreased from 367 to 293.

We have also done some tests where the node tasks are not included in the ATSP graph. Only the link tasks are included, which gives a smaller graph. The node tasks are then inserted correctly in the “re-tasking” phase. The reason for this is that we often ended up ignoring the positions of the node tasks, since they often were misplaced. This way we can save some time. The objective function value of the ATSP will then be too

Name	m	n	p	pg	low	an	aa	atim	obj	time
mini-01a	2	1	4	3	8	9	20	0.0	11	0.001872
mini-01b	2	1	4	4	10	9	18	0.0	11	0.002009
mini-01c	2	1	6	5	12	13	34	0.0	15	0.001610
mini-02a	3	2	7	5	12	15	56	0.0	17	0.001413
mini-02b	3	2	7	7	16	15	52	0.0	17	0.001409
mini-02c	3	2	9	7	16	19	82	0.0	21	0.001426
mini-03a	5	5	15	10	22	31	240	0.0	26	0.001527
mini-03b	5	5	15	12	25	31	236	0.0	31	0.001509
mini-03c	5	5	15	13	27	31	234	0.0	31	0.001474
mini-03d	5	5	15	13	27	31	234	0.0	31	0.001224
mini-03e	5	5	21	17	33	43	436	0.0	36	0.001548
mini-04a	8	9	26	17	34	53	702	0.0	40	0.001336
mini-04b	8	9	26	22	45	53	692	0.0	48	0.001650
mini-04c	8	9	26	26	50	53	684	0.0	58	0.001457
mini-05a	5	5	15	10	20	31	240	0.0	26	0.001133
mini-05b	5	5	15	15	28	31	230	0.0	29	0.001255
mini-05c	5	5	25	20	36	51	610	0.0	141	0.001310
mini-06a	9	11	31	20	37	63	992	0.0	43	0.002107
mini-06b	9	11	31	25	45	63	982	0.0	53	0.001910
mini-06c	9	11	49	40	68	99	2374	0.1	75	0.102939
mini-08a	9	15	39	24	38	79	1560	0.0	43	0.002011
mini-08b	9	15	39	34	52	79	1540	0.0	60	0.003357
mini-08c	9	15	39	34	52	79	1540	0.0	60	0.003601
mini-09a	18	26	70	44	72	141	4970	0.1	83	0.106022
mini-09b	18	26	70	70	104	141	4918	0.0	105	0.006184
mini-09c	18	26	70	70	104	141	4918	0.0	105	0.005931

Table 1: Results for problem set 1 with ZSRP.

Name	m	n	p	pg	low	an	aa	atim	obj	time
askeby	53	56	275	220	313	551	75458	3.5	367	3.5319
atvid-a-1	91	113	593	455	576	1187	351240	23.8	658	23.9737
atvid-a-2	71	96	455	359	450	911	206712	14.4	526	14.5206
atvid-a-3	34	49	230	181	217	461	52738	3.1	253	3.1352
bankekind	30	35	168	134	178	337	28118	1.3	213	1.3290
borensberg-b	103	131	605	485	634	1211	365648	23.3	741	23.5401
boxholm-1	90	119	570	449	567	1141	324514	23.8	663	23.9926
brokind-1	89	101	479	385	540	959	229154	14.2	621	14.3554
ekangen-1	74	82	380	309	433	761	144190	7.1	517	7.1987
mantorp-b	68	83	400	317	419	801	159736	9.2	480	9.3051
norsholm	64	77	368	293	377	737	135188	6.5	444	6.5715
rimforsa-a-1	64	74	354	283	391	709	125096	7.6	456	7.6498
rimforsa-a-2	30	34	158	128	178	317	24874	1.4	199	1.4177
skanninge-a-1	68	86	410	325	421	821	167828	11.5	491	11.6023
skanninge-a-2	45	55	263	209	264	527	68998	3.1	305	3.1555
ull1	41	43	199	163	240	399	39498	2.0	285	2.0290
ull2	44	47	226	182	254	453	50944	2.0	282	2.0258
valla-a	40	55	250	200	250	501	62340	3.0	293	3.0465
valla	69	85	393	316	415	787	154210	8.3	492	8.3711
vikingsstad-a-1	32	35	168	135	191	337	28124	1.4	219	1.4257
vikingsstad-a-2	63	86	407	321	404	815	165368	10.6	459	10.7122
vikingsstad-a-3	15	17	83	66	87	167	6836	0.3	96	0.3104

Table 2: Results for problem set 2 with ZSRP.

Name	m	n	p	pg	low	an	aa	atim	obj	time
mini-01a	2	1	4	3	8	9	20	0.0	11	0.001860
mini-01b	2	1	4	4	10	9	18	0.0	11	0.001418
mini-01c	2	1	4	4	10	9	18	0.0	11	0.001983
mini-02a	3	2	7	5	12	15	56	0.0	17	0.002106
mini-02b	3	2	7	7	16	15	52	0.0	17	0.001450
mini-02c	3	2	7	6	14	15	54	0.0	17	0.001402
mini-03a	5	5	15	10	22	31	240	0.0	26	0.001625
mini-03b	5	5	15	12	25	31	236	0.0	31	0.001849
mini-03c	5	5	15	13	27	31	234	0.0	31	0.002476
mini-03d	5	5	15	13	27	31	234	0.0	31	0.001587
mini-03e	5	5	15	14	28	31	232	0.0	31	0.001622
mini-04a	8	9	26	17	34	53	702	0.0	40	0.001800
mini-04b	8	9	26	22	45	53	692	0.0	48	0.002045
mini-04c	8	9	26	26	50	53	684	0.0	58	0.002002
mini-05a	5	5	15	10	20	31	240	0.0	26	0.001703
mini-05b	5	5	15	15	28	31	230	0.0	29	0.001532
mini-05c	5	5	15	15	28	31	230	0.0	29	0.001609
mini-06a	9	11	31	20	37	63	992	0.0	43	0.002469
mini-06b	9	11	31	25	45	63	982	0.0	53	0.002802
mini-06c	9	11	31	31	54	63	970	0.0	59	0.002100
mini-08a	9	15	39	24	38	79	1560	0.0	43	0.003813
mini-08b	9	15	39	34	52	79	1540	0.0	60	0.003930
mini-08c	9	15	39	34	52	79	1540	0.0	60	0.001844
mini-09a	18	26	70	44	72	141	4970	0.1	83	0.103490
mini-09b	18	26	70	70	104	141	4918	0.0	105	0.006189
mini-09c	18	26	70	70	104	141	4918	0.0	105	0.003620

Table 3: Results for problem set 1 with SRP.

Name	m	n	p	pg	low	an	aa	atim	obj	time
askeby	53	56	195	180	273	391	38018	0.7	293	0.7351
atvid-a-1	91	113	477	397	518	955	227352	11.8	551	11.9033
atvid-a-2	71	96	263	263	354	527	69240	0.5	355	0.5559
atvid-a-3	34	49	146	139	175	293	21322	0.3	184	0.3122
bankekind	30	35	128	114	158	257	16358	0.4	179	0.4098
borensberg-b	103	131	397	381	530	795	157648	2.8	545	2.8739
boxholm-1	90	119	360	344	462	721	129634	2.0	473	2.0728
brokind-1	89	101	311	301	456	623	96770	1.6	471	1.6474
ekangen-1	74	82	252	245	369	505	63550	1.1	388	1.1325
mantorp-b	68	83	256	245	347	513	65560	1.1	366	1.1332
norsholm	64	77	260	239	323	521	67580	1.4	338	1.4522
rimforsa-a-1	64	74	226	219	327	453	51112	0.8	335	0.8365
rimforsa-a-2	30	34	106	102	152	213	11250	0.2	159	0.2067
skanninge-a-1	68	86	256	248	344	513	65572	1.0	353	1.0579
skanninge-a-2	45	55	211	183	238	423	44454	1.3	263	1.3229
ull1	41	43	127	127	204	255	16170	0.1	207	0.1092
ull2	44	47	158	148	220	317	24968	0.4	231	0.4122
valla-a	40	55	152	151	201	305	23140	0.2	203	0.2131
valla	69	85	253	246	345	507	64050	0.9	353	0.9305
vikingsstad-a-1	32	35	102	102	158	205	10436	0.1	161	0.1067
vikingsstad-a-2	63	86	249	242	325	499	62036	0.8	340	0.8544
vikingsstad-a-3	15	17	53	51	72	107	2816	0.1	75	0.1026

Table 4: Results for problem set 2 with SRP.

Name	m	n	p	pg	low	an	aa	atim	obj	time
mini-01a	2	1	4	3	8	5	10	0.0	11	0.001618
mini-01b	2	1	4	4	10	5	8	0.0	11	0.001545
mini-01c	2	1	6	5	12	9	18	0.0	15	0.002005
mini-02a	3	2	7	5	12	9	28	0.0	17	0.001431
mini-02b	3	2	7	7	16	9	24	0.0	17	0.001554
mini-02c	3	2	9	7	16	13	44	0.0	21	0.001418
mini-03a	5	5	15	10	22	21	130	0.0	26	0.001577
mini-03b	5	5	15	12	25	21	126	0.0	31	0.001471
mini-03c	5	5	15	13	27	21	124	0.0	31	0.001471
mini-03d	5	5	15	13	27	21	124	0.0	31	0.001384
mini-03e	5	5	21	17	33	33	272	0.0	36	0.002008
mini-04a	8	9	26	17	34	37	378	0.0	40	0.001888
mini-04b	8	9	26	22	45	37	368	0.0	48	0.001627
mini-04c	8	9	26	26	50	37	360	0.0	51	0.001413
mini-05a	5	5	15	10	20	21	130	0.0	26	0.001533
mini-05b	5	5	15	15	28	21	120	0.0	29	0.001054
mini-05c	5	5	25	20	36	41	410	0.0	141	0.001634
mini-06a	9	11	31	20	37	45	550	0.0	44	0.001591
mini-06b	9	11	31	25	45	45	540	0.0	53	0.001809
mini-06c	9	11	49	40	68	81	1626	0.1	77	0.103507
mini-08a	9	15	39	24	38	61	990	0.0	43	0.002780
mini-08b	9	15	39	34	52	61	970	0.0	59	0.003248
mini-08c	9	15	39	34	52	61	970	0.0	59	0.003119
mini-09a	18	26	70	44	72	105	2860	0.1	82	0.105375
mini-09b	18	26	70	70	104	105	2808	0.0	105	0.002864
mini-09c	18	26	70	70	104	105	2808	0.0	105	0.005016

Table 5: Results for problem set 1 with ZSRP without node tasks in ATSP.

low, since the time for the node tasks are missing. However, the final solutions will be correct, with the correct objective function value.

Comparing tables 2 and 6, we find that the solutions times were decreased, in many cases approximately by a factor of two. The solutions, represented by the objective function value, were quite similar, although with some slight variations, which we believe are random effects of the heuristics.

Comparing tables 4 and 8, we find the same kind of differences, lower solutions times, and in principle the same solutions. For the first example, askeby, the solution time was decreased to 0.41 seconds.

It is thus clear that it is possible to save time by omitting the node tasks in the ATSP graph, and add them afterwards.

We draw the following conclusions of the tests. The mini-problems are all solved very quickly, the maximal time was around 0.1 second, and are too easy to give significant differences between the different variations.

Name	m	n	p	pg	low	an	aa	atim	obj	time
askeby	53	56	275	220	313	445	49398	1.7	364	1.7401
atvid-a-1	91	113	593	455	576	1005	252232	11.6	664	11.7379
atvid-a-2	71	96	455	359	450	769	147648	7.0	529	7.0987
atvid-a-3	34	49	230	181	217	393	38514	1.5	253	1.5262
bankekind	30	35	168	134	178	277	19116	0.7	202	0.7113
borensberg-b	103	131	605	485	634	1005	252288	10.6	740	10.7702
boxholm-1	90	119	570	449	567	961	230642	11.0	645	11.1548
brokind-1	89	101	479	385	540	781	152316	6.6	626	6.7051
ekangen-1	74	82	380	309	433	613	93822	3.4	504	3.4619
mantorp-b	68	83	400	317	419	665	110390	4.8	473	4.8752
norsholm	64	77	368	293	377	609	92574	3.3	425	3.3553
rimforsa-a-1	64	74	354	283	391	581	84254	4.4	444	4.4527
rimforsa-a-2	30	34	158	128	178	257	16460	0.5	201	0.5100
skanninge-a-1	68	86	410	325	421	685	117138	5.4	490	5.4538
skanninge-a-2	45	55	263	209	264	437	47636	1.7	315	1.7309
ull1	41	43	199	163	240	317	25064	0.8	283	0.8162
ull2	44	47	226	182	254	365	33224	1.3	281	1.3295
valla-a	40	55	250	200	250	421	44220	2.0	296	2.0248
valla	69	85	393	316	415	649	105162	4.3	485	4.3763
vikingsstad-a-1	32	35	168	135	191	273	18570	0.6	213	0.6163
vikingsstad-a-2	63	86	407	321	404	689	118508	5.6	455	5.6589
vikingsstad-a-3	15	17	83	66	87	137	4658	0.1	103	0.1060

Table 6: Results for problem set 2 with ZSRP without node tasks in ATSP.

Name	m	n	p	pg	low	an	aa	atim	obj	time
mini-01a	2	1	4	3	8	5	10	0.0	11	0.001972
mini-01b	2	1	4	4	10	5	8	0.0	11	0.001745
mini-01c	2	1	4	4	10	5	8	0.0	11	0.001728
mini-02a	3	2	7	5	12	9	28	0.0	17	0.001709
mini-02b	3	2	7	7	16	9	24	0.0	17	0.001651
mini-02c	3	2	7	6	14	9	26	0.0	17	0.001860
mini-03a	5	5	15	10	22	21	130	0.0	26	0.001995
mini-03b	5	5	15	12	25	21	126	0.0	31	0.001798
mini-03c	5	5	15	13	27	21	124	0.0	31	0.002181
mini-03d	5	5	15	13	27	21	124	0.0	31	0.002047
mini-03e	5	5	15	14	28	21	122	0.0	31	0.001890
mini-04a	8	9	26	17	34	37	378	0.0	40	0.001819
mini-04b	8	9	26	22	45	37	368	0.0	48	0.001265
mini-04c	8	9	26	26	50	37	360	0.0	51	0.001295
mini-05a	5	5	15	10	20	21	130	0.0	26	0.001041
mini-05b	5	5	15	15	28	21	120	0.0	29	0.001314
mini-05c	5	5	15	15	28	21	120	0.0	29	0.001437
mini-06a	9	11	31	20	37	45	550	0.0	44	0.001331
mini-06b	9	11	31	25	45	45	540	0.0	53	0.001680
mini-06c	9	11	31	31	54	45	528	0.0	55	0.001701
mini-08a	9	15	39	24	38	61	990	0.0	43	0.002931
mini-08b	9	15	39	34	52	61	970	0.0	59	0.003598
mini-08c	9	15	39	34	52	61	970	0.0	59	0.003182
mini-09a	18	26	70	44	72	105	2860	0.1	82	0.104955
mini-09b	18	26	70	70	104	105	2808	0.0	105	0.003552
mini-09c	18	26	70	70	104	105	2808	0.0	105	0.005215

Table 7: Results for problem set 1 with SRP without node tasks.

Name	m	n	p	pg	low	an	aa	atim	obj	time
askeby	53	56	195	180	273	285	20358	0.4	297	0.41495
atvid-a-1	91	113	477	397	518	773	149340	5.3	545	5.41591
atvid-a-2	71	96	263	263	354	385	37248	0.1	363	0.13000
atvid-a-3	34	49	146	139	175	225	12726	0.2	183	0.20948
bankekind	30	35	128	114	158	197	9716	0.2	175	0.21258
borensberg-b	103	131	397	381	530	589	86928	1.2	545	1.28132
boxholm-1	90	119	360	344	462	541	73352	1.0	469	1.03462
brokind-1	89	101	311	301	456	445	49668	0.7	470	0.73935
ekangen-1	74	82	252	245	369	357	31998	0.4	385	0.42451
mantorp-b	68	83	256	245	347	377	35654	0.6	359	0.63603
norsholm	64	77	260	239	323	393	38682	0.7	345	0.71900
rimforsa-a-1	64	74	226	219	327	325	26526	0.3	335	0.32760
rimforsa-a-2	30	34	106	102	152	153	5904	0.1	158	0.10991
skanninge-a-1	68	86	256	248	344	377	35672	0.5	355	0.52217
skanninge-a-2	45	55	211	183	238	333	27720	0.7	259	0.71680
ull1	41	43	127	127	204	173	7568	0.0	205	0.01219
ull2	44	47	158	148	220	229	13164	0.2	231	0.21698
valla-a	40	55	152	151	201	225	12762	0.1	203	0.10770
valla	69	85	253	246	345	369	34182	0.4	353	0.42650
vikingsstad-a-1	32	35	102	102	158	141	5040	0.0	159	0.00840
vikingsstad-a-2	63	86	249	242	325	373	34926	0.4	333	0.43353
vikingsstad-a-3	15	17	53	51	72	77	1508	0.0	75	0.00255

Table 8: Results for problem set 2 with SRP without node tasks in the ATSP.

The city instances took longer, but never more than 30 seconds for ZSRP including node tasks in the ATSP. Almost all the time was spent in the ATSP solver, where problems with up to 1200 nodes and 265 000 arcs were solved.

Comparing the ZSRP with the SRP, we find several significant differences. The solution time is reduced significantly, when omitting the third sweep. A reduction factor of 10 is common. For the example atvid-a-1, the number of nodes was reduced from 1187 to 955, but the solution time decreased from 24 to 12 seconds, while for atvid-a-2, the number of nodes went from 911 to 527, and the solution time from 14.5 to 0.6 seconds.

Also the optimal objective function value shows a clear difference. In the two example mentioned above, from 658 to 551, and from 526 to 355. It is clearly more expensive to use a third sweep, i.e. it takes longer before all streets are cleared completely.

Omitting the node tasks in the ATSP solution gives shorter solution times. For the most difficult instance, atvid-a-1, the solution time was decreased to 5.4 seconds. while all other instances were solved in around one second or less.

8 Conclusions

We have constructed a solution method for the (over) zealous snow remover problem, based on a reformulation to the asymmetrical traveling salesman problem, combined with a reordering procedure and a solution finding procedure. Realistic test problems have been solved in quite reasonable times.

We find that the over zealous snow remover is faced with a more difficult optimization problem, both in theoretical and practical terms, than the normal one. The solutions are clearly more expensive. In practical terms, one should consider carefully if the middle sweep is motivated. On the other hand, the problem is not impossible to solve.

It is hard to measure the practical advantages of the middle sweep. In case of a severe snow storm, it could be impossible for cars to use the streets before the first middle sweep, and then it is well motivated. Otherwise it is probably not motivated. We note that it actually delays the point in time when everything is finished.

For these reasons, we do not give any general advice about the middle sweep. In this paper, we instead make a quantitative comparison of the two cases, in order to give some decision support for the contractors that have to make the decisions in practice.

References

- Helsgaun, K. (2000), “An effective implementation of the Lin-Kernighan traveling salesman heuristic”, *European Journal of Operational Research* 126, 106–130.
- Holmberg, K. (2014), “Urban snow removal: Modeling and relaxations”, Research report

LiTH-MAT-R-2014/08-SE, Department of Mathematics, Linköping University, Sweden.

Holmberg, K. (2015a), “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.

Holmberg, K. (2015b), “On using OpenStreetMap and GPS for optimization”, Research report LiTH-MAT-R-2015/15-SE, Department of Mathematics, Linköping University, Sweden.

Noon, C. E., and Bean, J. C. (1993), “An efficient transformation of the generalized traveling salesman problem”, *INFOR* 31, 39–44.

Perrier, N., Langevin, A., and Amaya, C.-A. (2008), “Vehicle routing for urban snow plowing operations”, *Transportation Science* 42, 44–56.

Perrier, N., Langevin, A., and Campbell, J. F. (2007), “A survey of models and algorithms for winter road maintenance: Part IV: Vehicle routing and fleet sizing for plowing and snow disposal”, *Computers and Operations Research* 34, 258 – 294.

Soler, D., Martinez, E., and Mico, J. (2008), “A transformation for the mixed general routing problem with turn penalties”, *Journal of the Operational Research Society* 59, 540–547.